



UNIVERSITATEA *TRANSILVANIA* DIN BRAȘOV

Centrul de Învățământ la Distanță
și Învățământ cu Frecvență Redusă



FACULTATEA DE MATEMATICĂ ȘI INFORMATICĂ
PROGRAM DE LICENȚĂ: INFORMATICĂ

Arhitectura calculatoarelor

CURS PENTRU ÎNVĂȚĂMÂNT LA DISTANȚĂ

AUTORI: Lect. Dr. Anca Vasilescu

2012-2013



CUVÂNT ÎNAINTE



În societatea actuală, în viața de zi cu zi, suntem din ce în ce mai mult dependenți de calculator. Această *mașină* pe care ne dorim s-o stăpânim cât mai bine este în măsură să facă în locul nostru o mulțime de operații. În acest sens, principalul avantaj este câștigarea de timp liber datorată vitezei cu care calculatorul ne rezolvă problemele. Pentru a putea valorifica un sistem de calcul la adevărata lui valoare, un utilizator trebuie să știe strictul necesar despre structura și funcționarea calculatorului. Ulterior, pe baza acestui vocabular minimal, el poate să aprofundeze un anumit domeniu, o anumită direcție de folosire a calculatorului, în directă concordanță cu propria sa activitate cotidiană.

Acest curs își propune tocmai acest **obiectiv final**: să familiarizeze utilizatorul începător cu principalele componente ale calculatorului, cu structura lor de bază și rolul fiecăreia în sistem.

Materialul didactic scris pentru studenți este împărțit în: (I) suport pentru activitățile tutoriale – care conține considerentele teoretice corespunzătoare capitolelor 1 și 3 din tematica cursului și (II) suport pentru activitățile aplicative – care conține considerentele teoretice și problemele rezolvate corespunzătoare capitolului 2 din tematica cursului.

Bibliografia propusă în acest suport de curs este una selectivă. Pentru pregătirea laboratoarelor și a temelor individuale, studenții pot folosi orice material care tratează tema în discuție și care este potrivit nivelului de pregătire al fiecăruia.

Termenele de predare a temelor individuale/temelor de casă se vor stabili de comun acord cu studenții prezenți la activitățile tutoriale și aplicative. Criteriile de evaluare a materialelor predate ca temă de casă sunt: încadrarea în subiect, originalitatea, bibliografia. Studenții vor consulta cel puțin trei referințe bibliografice pentru realizarea fiecărei teme de casă.

Opțional, studenții sunt invitați să realizeze practic circuite electronice pentru simularea funcționării circuitelor logice interne ale componentelor calculatorului. Pentru această temă, studenții interesați vor lua legătura direct cu cadrul didactic.



OBIECTIVE

Acest curs își propune următoarele obiective educaționale:

1. Însușirea cunoștințelor de bază referitoare la un sistem de calcul.
2. Familiarizarea cu arhitectura calculatoarelor personale IBM-PC.
3. Familiarizarea cu structura și modul de funcționare ale principalelor componente ale unui sistem de calcul.
4. Deprinderea programării în limbaj de asamblare.



NOTA FINALĂ

Nota finală rezultă ca medie ponderată între notele rezultate din: tratarea părții teoretice la lucrarea finală, rezolvarea problemelor din lucrarea finală, evaluarea altor activități din timpul tutorialelor și laboratoarelor. Activitățile opționale și suplimentare pot compensa greșelile de la tratarea lucrării finale.



TEME DE CASĂ

TC1. De la ABAC la calculatorul cuantic. Evoluția arhitecturii mașinilor de calcul

TC2. Pachet de soluții pentru descrierea la nivel logic –digital a unor componente elementare ale calculatorului



TEMATICA CURSULUI

⇒ Componentele de bază ale unui sistem de calcul

✎ Noțiuni introductive

✎ Definiții. SC – din diferite puncte de vedere. Structura unui sistem de calcul. Performanțe. Clasificări. Principiile von Neumann. Modelul formal al unui calculator.

✎ Sistemul de intrare/ ieșire. Dispozitive periferice.

✎ Clasificări. Canalul de intrare/ ieșire. Legarea perifericelor la SC. Procesoare de intrare/ ieșire. Structura unor clase de periferice.

✎ Unitatea de memorie

✎ Definiții. Structura fizică. Structura ierarhică de organizare a memoriei. Tipuri de acces la memorii.

⇒ Abordarea la nivel logic-digital a componentelor calculatorului

✎ Circuite logice

✎ Porți logice. CLC. CLS. CLC aritmetice. CLC decodare. CLC multiplexoare. CLS bistabile. Circuite integrate.

✎ Regiștri

✎ Regiștri cu încărcare paralelă. Regiștri de deplasare. Numărătoare binare. Regiștri de incrementare.

✎ Structura de bază a unui calculator

✎ Regiștri de bază ai unui sistem de calcul. Unitatea de contorizare și control. Transferul datelor pe magistrală.

⇒ Unitatea centrală de prelucrare

✎ Organizarea generală a regiștrilor. Organizarea stivei

✎ Microoperație. Instrucțiune. Program. Codurile instrucțiunilor

✎ Moduri de adresare

✎ Clasificarea instrucțiunilor

✎ Instrucțiuni pentru transferul datelor. Instrucțiuni pentru prelucrarea datelor. Instrucțiuni pentru controlul programului.

✎ Prelucrarea instrucțiunilor unui program

✎ Microprocesorul

✎ Caracteristici. Setul de instrucțiuni. Modelarea funcționării unui procesor. Mașina cu trei adrese. Mașina cu o adresă. Structura internă de bază a unui microprocesor I 80x86.

✎ Introducere în utilizarea limbajului de asamblare. Sintaxa principalelor instrucțiuni. Rezolvarea unor probleme elementare.

✎ Multiprocesare. Coprocesoare matematice.



TEMATICA SEMINARULUI

- AT1. Clasificarea resurselor fizice ale calculatoarelor moderne. Clasificarea resurselor logice ale calculatoarelor moderne.
- AT2. Calculul performanțelor principalelor componente ale sistemelor de calcul moderne
- AT3. Modelarea funcționării circuitelor logice. Probleme cu CLC, CLS
- AT4. Modelarea funcționării circuitelor logice. Probleme cu CBB, registri
- AT5. Descrierea la nivel logic-digital a structurii principalelor componente interne ale calculatoarelor moderne



TEMATICA LABORATORULUI

- AA1. Caracterizarea arhitecturii sistemelor de calcul moderne ale anului 2008
- AA2. Concepte de baza in dezvoltarea calculatoarelor viitorului. Sisteme nesecventiale
- AA3. Elemente de utilizare a unui produs software pentru editarea si simularea functionarii circuitelor logice
- AA4. Simularea funcționării componentelor interne elementare ale calculatorului folosind un produs software de editare si simulare a circuitelor logice
- AA5. Simularea funcționării unor componente interne complexe ale calculatorului folosind un produs software de editare si simulare a circuitelor logice



BIBLIOGRAFIE GENERALĂ

- [1] JOHN von NEUMANN
The Computer and the Brain, Yale University Press, New Haven, Conn. 1958
- [2] MORRIS MANO M.
Computer System Architecture, Prentice Hall International, 1993
- [3] BOIAN F.M.
De la aritmetică la calculatoare, Presa Universitară Clujeană, 1996
- [4] MUSCA G.
Programare în limbaj de asamblare, Editura Teora, București, 1998
- [5] PATTERSON D.A., HENNESSY J.L.
Computer Organization and Design – The Hardware / Software Interface, Morgan Kaufmann Publishers, Inc., 1998 tradusă și în românește la Editura ALL, 2002
- [6] BURILEANU C. ș.a.
Microprocesoarele x86 – o abordare software, Casa de Editură Albastră, Grupul Microinformatica, Cluj-Napoca, 1999
- [7] TANENBAUM A.S.
Organizarea structurată a calculatoarelor, Computer Libris Agora, Cluj-Napoca, 1999 traducere după *Structured Computer Organization*, Prentice Hall International, 1990
- [8] CHIOREAN L. ș.a.
PC – inițiere hard și soft, Casa de Editură Albastră, Grupul Microinformatica, Cluj-Napoca, 1999
- [9] ENGLANDER I.
The Architecture of Computer Hardware and Systems Software, John Wiley & Sons, Inc. 2000
- [10] STEFAN GH. ș.a.
Circuite integrate digitale – probleme proiectare, Casa de Editură Albastră, Grupul Microinformatica, Cluj-Napoca, 2000
- [11] BARUCH Z .F.
Sisteme de intrare/ieșire ale calculatoarelor, Casa de Editură Albastră, Grupul Microinformatica, Cluj-Napoca, 2000
- [12] GORGAN D. ș.a.
Structura calculatoarelor, Casa de Editură Albastră, Grupul Microinformatica, Cluj-Napoca, 2000
- [13] LUNGU V.
Procesoare INTEL. Programare în Limbaj de asamblare, Editura Teora, București, 2000
- [14] VASILESCU A.
Concepte de bază în tehnologia informației, Editura Universității din Pitești, 2000
- [15] EDER B., ș.a.
ABC-ul calculatoarelor, volumul 1 în seria Computer Driving Licence, Editura BIC ALL, București, 2001
- [16] Reviste de specialitate



COMPONENTELE DE BAZĂ ALE SISTEMELOR DE CALCUL

CUPRINS

1.1.	Sistemul de calcul	7
1.1.1	Definiții	7
1.1.2	Sistemul de calcul – din diferite puncte de vedere	7
1.1.3	Structura ierarhică de organizare a calculatorului	8
1.1.4	Structura funcțională a unui sistem de calcul	9
1.1.5	Principiile von Neumann	11
1.1.6	Evoluția sistemelor de calcul	11
1.2.	Sistemul de intrare/ieșire. Dispozitive periferice	15
1.2.1	Clasificări	15
1.2.2	Caracterizarea unor periferice	16
1.2.2.1	Discurile dure. Hard-discul	16
1.2.2.2	Discurile flexibile. Discheta	17
1.2.2.3	Compact discurile	17
1.2.2.4	Monitoarele (<i>graphic display</i>)	17
1.2.2.5	Mouse-ul	18
1.2.2.6	Rețele de comunicare	19
1.2.2.7	Placa de rețea	19
1.2.2.8	Modemul	19
1.2.3	Legarea perifericelor la SC. Unitățile de interfață I/O	20
1.2.4	Magistrale	21
1.2.4.1	Clasificarea magistrelor	23
1.2.4.2	Standarde de magistrală	25
1.3.	Unitatea de memorie	27
1.3.1	Structura fizică a memoriei	27
1.3.2	Tehnologii de realizare a memoriilor. Memorii semiconductoare	28
1.3.3	Structura ierarhică de organizare a memoriei	30
1.3.4	Memorii cache	33
1.4.	Proiectarea calculatoarelor moderne	34

1.1. Sistemul de calcul

1.1.1 Definiții

Sistemul de calcul (SC, *System Computer* sau calculator) este reprezentat de o structură destinată prelucrării datelor.

Adesea, un SC este descris prin cele două subsisteme componente: **hardware** și sistemul **software**.

Partea **hardware** constă din toate componentele electronice care alcătuiesc partea fizică a SC. Sistemul **software** constă din instrucțiuni și date pe care SC le prelucrează pentru a executa diversele cerințe ale utilizatorului. O secvență de instrucțiuni ale SC se numește **program**. Datele prelucrate de SC constituie **baza de date**.

Totalitatea componentelor sistemului de calcul solicitate de programe pentru executarea lor poartă numele de **resurse**.

Concret, **SC** este format din: resurse fizice, resurse logice și resurse informaționale. Avem următoarele corespondențe:

1. **hardware = resurse fizice**, adică procesoare, memorii, dispozitive de intrare/ieșire, ș.a. O clasificare acoperitoare a resurselor fizice poate porni de la gradul de răspândire (utilizare) a unei componente fizice a calculatorului la momentul la care se face clasificarea. Rezultă astfel: (a) resurse fizice care formează un sistem de calcul minimal, (b) resurse fizice care formează un sistem de calcul uzual și (c) resurse fizice care formează un sistem de calcul specializat.
2. **software = resurse logice**. Acestea se împart în (a) software pentru sistem (*systems software*) și (b) software pentru aplicații (*applications software*). **Soft-ul pentru sistem** constă în aplicațiile sistem care oferă servicii pentru uz comun, în special pentru gestionarea sistemului însuși și, eventual, a rețelei/rețelelor din care acesta face parte. Din această categorie fac parte: sistemul de operare, drivere, servere ș.a. **Soft-ul pentru aplicații** constă în aplicațiile propriu-zise, adică: aplicații și/sau medii de dezvoltare și programare, aplicații utilitare, aplicații utilizator. Aplicațiile sistem sunt la granița dintre hardware și aplicațiile propriu-zise.

Observație. Trebuie precizat că această clasificare este una relativă deoarece un anumit program nu este obligatoriu să facă parte exclusiv dintr-o astfel de categorie. De exemplu, un compilator este o aplicație sistem, dar, mai mult, poate face parte dintr-un sistem de operare, dacă este vândut împreună cu acesta și depinde de acel sistem de operare.

Un sistem de operare este un set de programe care gestionează toate resursele sistemului de calcul gazdă, astfel încât programele care rulează pe acel sistem să se execute la parametri maximi (utilizare completă și optimă).

3. **resurse informaționale**, adică fișiere de date și fișierele utilizatorilor.

1.1.2 Sistemul de calcul – din diferite puncte de vedere

Descrierea operațiilor unui SC la nivel de *hardware* diferă în funcție de scopul urmărit. Astfel, un SC poate fi privit cel puțin din trei puncte de vedere:

- ✓ organizarea componentelor;
- ✓ construirea;
- ✓ arhitectura internă.

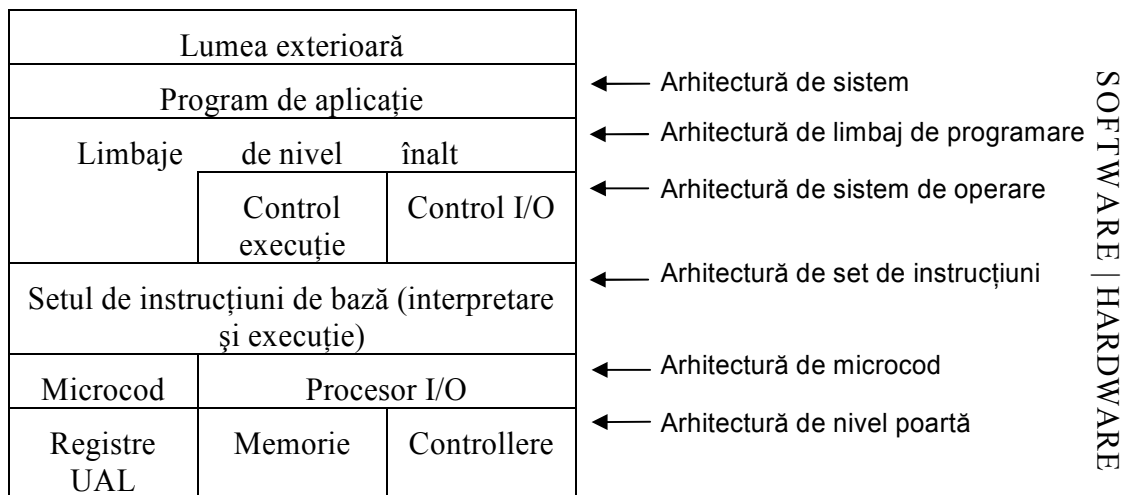
Organizarea SC se referă la cunoașterea modului în care operează fiecare dintre componentele hard și posibilitățile lor de interconectare astfel încât să formeze SC performant.

Construirea SC (design) se referă la acțiunea de determinare a tipului componentelor hard care trebuie folosite și a tipurilor de interconectare a acestora, astfel încât SC construit să răspundă cerințelor utilizatorului.

Arhitectura sistemului de calcul se referă la structura și comportarea SC văzut de utilizator. Când ne referim la arhitectura unui SC trebuie să dăm specificațiile diferitelor unități funcționale și conectările fizice și logice dintre ele.

Conform DEX (Dicționarului Explicativ al limbii române), **arhitectura calculatoarelor** se referă la asamblarea subsistemelor funcționale într-un calculator numeric, pentru a obține performanțele impuse în condiții de cost avantajoase.

Conform [13], sintagma *arhitectura calculatoarelor* este adesea utilizată cu semnificația simplă de *organizarea și proiectarea calculatoarelor*. Detaliind din punct de vedere practic, într-un sistem de calcul se pot distinge mai multe nivele de arhitectură (vezi figura următoare), fiecare definit de legătura pe care o stabilește între subsistemele sistemului gazdă.



În general, prin *arhitectură de nivel* ne referim la interfața dintre două astfel de module funcționale. **Arhitectura de sistem** sau **arhitectura sistemului de calcul** este cea dintre sistemul de calcul și mediul exterior lui. La limita dintre *hardware* și *software* se află arhitectura setului de instrucțiuni care reprezintă nivelul elementar la care sunt decodificate și executate instrucțiunile recunoscute de unitatea de prelucrare.

1.1.3 Structura ierarhică de organizare a calculatorului

Din punctul de vedere al rolului fiecărei resurse a sistemului de calcul putem privi organizarea internă a calculatoarelor moderne pe două sau mai multe nivele. Fiecare nivel este construit pe baza predecesorului său și reprezintă o abstractizare distinctă, cu obiecte și operații specifice.

În general, se pot identifica următoarele șase nivele conceptuale:

Nivelul 0. Nivelul logic digital. La acest nivel interesează **porțile logice** care intră în structura fiecărei componente interne a calculatorului. Fiecare poartă este construită dintr-un număr mic de tranzistoare. Câteva porți pot fi combinate pentru a forma un bistabil (o memorie de 1 bit), care poate stoca 0 sau 1. Memoriile de 1 bit pot fi combinate pentru a forma regiștri. În acest curs, nivelul logic digital este subiectul capitolului al doilea.

Nivelul 1. Nivelul microarhitecturii. La acest nivel calculatorul este o colecție de **regiștri** (care formează memoria locală) și un circuit capabil să execute operații aritmetice și logice (numit UAL, **unitate aritmetico-logică**). Regiștri sunt conectați la UAL pentru a forma o cale de date (engl. *datapath*) prin care se transferă datele între regiștrii și UAL. Operația de bază la nivelul căii datelor este selecția regiștrilor asupra cărora operează UAL. După felul în care este controlat transferul datelor deosebim mașini de calcul cu control microprogramat sau mașini de calcul cu control bazat și pe circuite ale componentelor hardware.

Nivelul 2. Nivelul arhitecturii setului de instrucțiuni sau nivelul ISA (engl. *Instruction Set Architecture*). Fiecare producător de calculatoare publică un manual pentru mașinile pe care le vinde, manual care se referă la acest nivel ISA, dar nu și la nivelele de dedesubt. În acest manual producătorul descrie **limbajul mașinii** pe care o vinde, adică setul de instrucțiuni mașină recunoscute de mașina respectivă și pe care, implicit, aceasta le poate executa. Această descriere constă în detalierea instrucțiunilor executate în mod interpretat de către componenta de control (microprogram sau circuite hardware).

Nivelul 3. Nivelul mașină al sistemului de operare. La acest nivel se detaliază structura și funcțiile sistemului de operare al mașinii. Față de nivelul ISA, sunt adăugate instrucțiuni noi, există o organizare diferită a memoriei, se remarcă posibilitatea execuției paralele și/sau concurente a programelor (task-urilor).

Observație. În general, nivelele de la 1 la 3 nu sunt proiectate pentru a fi utilizate de programatorul obișnuit. Ele sunt un suport pentru aplicațiile dezvoltate la nivele superioare de către programatorii de aplicații. O altă diferență importantă între nivelele 1-3 pe de o parte și nivelele 4,5 pe de altă parte este natura limbajului oferit. Limbajele mașină ale nivelelor inferioare sunt numerice (programele sunt secvențe greoaie de numere binare), în timp ce programele dezvoltate la nivelele superioare folosesc limbaje care conțin cuvinte uzuale sau abrevieri ușor de înțeles.

Nivelul 4. Nivelul limbajului de asamblare. Acest nivel oferă programatorilor posibilitatea de a scrie programe pentru nivelele inferioare într-o formă simbolică, mai accesibilă decât formatul impus de limbajul mașină.

Nivelul 5. Nivelul limbajului orientat pe problemă. La acest nivel sunt proiectate limbaje pe care le vor folosi programatorii de aplicații care au de rezolvat probleme specifice. Astfel de limbaje sunt uzual numite **limbaje de nivel înalt**.

Mulțimea de tipuri de date, operații și caracteristici ale fiecărui nivel se numește **arhitectura** nivelului respectiv. Cu alte cuvinte, arhitectura se referă la acele aspecte care sunt vizibile pentru utilizatorul unui nivel.

Conform Tanenbaum [7] **arhitectura calculatoarelor** se referă la studiul proiectării acelor părți ale unui sistem de calcul care sunt vizibile programatorului. În general, arhitectura calculatoarelor și organizarea calculatoarelor sunt sintagme sinonime.

1.1.4 Structura funcțională a unui sistem de calcul

Într-un SC, resursele fizice împreună cu cele logice cooperează pentru satisfacerea cerințelor utilizatorilor în ceea ce privește: *introducerea* (recepționarea) datelor, *memorarea* (conservarea) datelor și informațiilor, *prelucrarea* informațiilor, *transmiterea* informațiilor la alte sisteme de calcul și *regăsirea* informațiilor.

Îndeplinirea acestor operații cade în sarcina unor subsisteme, numite **unități funcționale** ale SC. Acestea sunt conectate fizic și logic între ele și se individualizează prin funcția specifică fiecăreia în sistemul de calcul.

După funcțiile pe care le îndeplinesc, unitățile funcționale se grupează în următoarele clase, formând structura fizică a sistemului de calcul:

- **unități de schimb** – pentru recepționare și transmitere de informație. Unitățile de schimb formează **componenta de control și comandă (UCC)**.
- **componentele unității aritmetico-logice (UAL)** – pentru executarea operațiilor aritmetice și logice. Acestea preiau din memorie valorile operanzilor și depun tot în memorie rezultatele.
- **procesoare** – pentru prelucrarea datelor. Procesoarele din SC formează nucleul pentru **unitatea centrală de prelucrare (UCP sau CPU = Central Processing Unit)**. Microprocesorul denumește o unitate centrală de prelucrare realizată într-un singur circuit integrat.
- **memorii interne** – pentru stocarea datelor și instrucțiunilor;

- **dispozitive periferice** – pentru preluarea și/sau transmiterea informațiilor externe. Perifericele formează **sistemul de intrare/ieșire**.

Cu aceste componente se pot scrie ecuațiile care reprezintă structura logică a sistemului de calcul. Acestea sunt:

$$UCC + UAL = UCtrlP$$

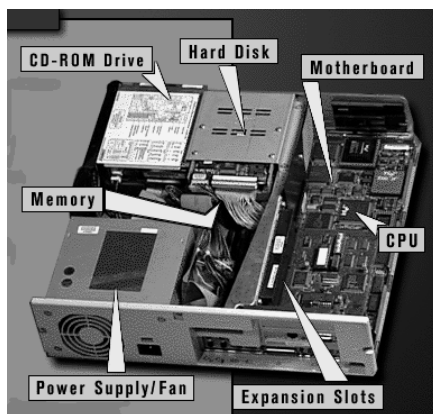
$$UCtrlP + \text{Regiștrii} = UCP$$

$$UCP + \text{Memorii} = UC$$

$$UC + \text{IOS} = SC$$

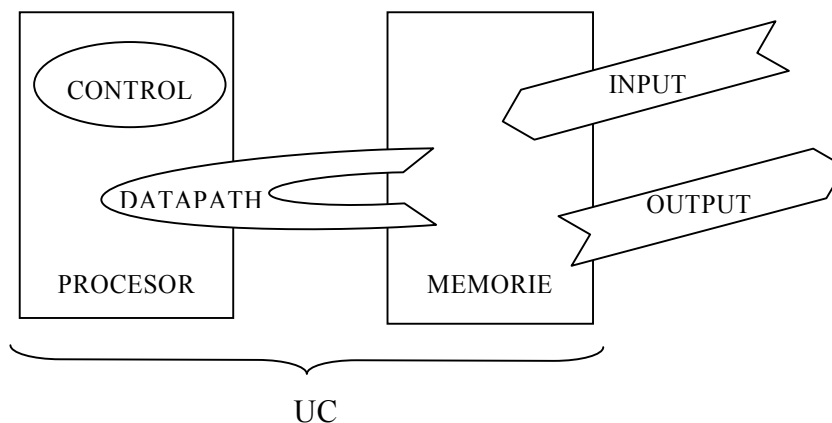
- ✓ **UCtrlP** este **unitatea de control și prelucrare**
- ✓ **UC** este **unitatea centrală**
- ✓ **IOS** este **sistemul de intrare/ieșire**.

Primele trei ecuații definesc *box-ul* calculatorului, în timp ce ultima ecuație definește legătura acestuia cu mediul exterior lui. UC este unitatea de control a întregului sistem de calcul: supervizează activitatea oricărei componente și decide în situații conflictuale. Aceste componente se regăsesc și în figura următoare:



Orice componentă a sistemului de calcul poate fi încadrată în una dintre categoriile: MEMORII, INPUT, OUTPUT, CONTROL, DATAPATH. Relațiile de dependență între unitățile funcționale ale SC vor fi detaliate ulterior, în paragraful referitor la *etapele prelucrării instrucțiunilor unui program*. Prezintă în figura următoare numai schema orientativă a acestor dependențe.

De aici rezultă că PROCESORul preia instrucțiunile și datele din memorie, în timp ce CONTROLul trimite semnale care determină operații pe DATAPATH, în MEMORII și pe dispozitivele periferice. Datapath (calea datelor) include registre, UAL și mai multe magistrale de legătură (vezi și exemplul de transfer al datelor pe Datapath din capitolul al treilea). Cu alte cuvinte, *datapath* este acea parte a UCP care conține UAL și intrările și ieșirile acesteia.



1.1.5 Principiile von Neumann

Incepând cu 1940, după apariția lucrărilor teoretice privind calculele automate, lucrări datorate în principal lui Alan Turing, lumea cercetătorilor a devenit tot mai preocupată de construcția efectivă a unor calculatoare. Matematicianul american John von Neumann a intuit, prin analogie cu anatomia creierului uman, principiile care ar trebui să stea la baza construcției calculatoarelor.

Enumerăm aceste principii, așa cum au fost formulate la vremea respectivă.

1. Deoarece mașina este destinată calculelor, ea execută cele patru operații aritmetice. Se impune deci, existența unei componente specializate pentru calcule. (Astăzi, aceasta este numită *unitatea aritmetico-logică*.)

2. Operațiile se vor executa secvențial. Pentru a asigura elasticitatea și o relativă universalitate, se impune o distincție între instrucțiunile necesare rezolvării unei probleme particulare și controlul general asupra unor instrucțiuni executate la un moment dat. (Apare, astfel, ceea ce astăzi numim *componentă de control*.)

3. Este necesară o componentă numită *memorie internă* în care „se țin minte” pentru un timp limitat atât instrucțiunile, cât și datele necesare rezolvării problemei.

Cele trei componente, luate împreună, sunt analoage *neuronilor* din sistemul nervos central.

Se impune, continuând analogia, să existe un senzor care să recepționeze semnalele de orice fel provenite din exterior. Este necesar, de asemenea, un motor care să acționeze asupra mediului exterior. (Astăzi, acestea se numesc *dispozitive de intrare/ieșire*.)

4. Este necesară o componentă care să „țină minte permanent”, să înregistreze așa cum omul își notează ceea ce nu dorește să uite. Această componentă o vom numi *memorie permanentă*. (Astăzi, ne referim la această componentă prin *memorie externă*.)

Este necesară crearea posibilității de trecere de la memoria internă la cea permanentă și reciproc.

Conform acestor principii, **arhitectura von Neumann** reprezintă structura ierarhică de organizare a unui calculator construit pe baza modelului von Neuman și care conține: o unitate de memorie principală, o unitate centrală de prelucrare, o cale internă pentru transferul datelor și dispozitive de intrare - ieșire.

Cu câteva excepții, și calculatoarele actuale sunt construite pe baza acestor principii. Totuși, **sistemele paralele**, numite și arhitecturi non von Neumann, nu respectă principiul secvențialității (2), în timp ce **sistemele simbolice**, utilizate în aplicații complexe din domeniul inteligenței artificiale, se bazează pe inferențe logice, deci nu exclusiv pe calcule aritmetice (ca în principiul 1).

Opinia multor cercuri științifice este că, în deceniul acesta, cele trei direcții fundamentale de cercetare care vor prezenta interes vor fi: microprocesoarele, inteligența artificială și prelucrarea distribuită. Se preconizează astfel o orientare preponderent spre **arhitecturile non von Neumann**, mai noi și mai puțin studiate...

1.1.6 Evoluția sistemelor de calcul

„De unde... un ENIAC este echipat cu 18000 de tuburi electronice și cântărește 30 tone, înseamnă că computerele viitorului ar putea avea 1000 tuburi și, probabil, o greutate de 1,5 tone. ”

Popular Machine, martie 1949

Deși este dificil de prezis performanțele viitoarelor SC, este sigur că vor depăși cu mult caracteristicile celor actuale.

Atât creatorii de soft cât și cei de hard construiesc sistemele de calcul bazându-se pe structura ierarhică de organizare în care fiecare nivel cumulează caracteristicile nivelelor inferioare.

Tehnologiile cheie pentru procesoarele moderne sunt compilatoarele și siliconul. În timp ce siliconul permite mărirea capacităților hard, noile idei în organizarea calculatoarelor îmbunătățesc considerabil raportul preț / performanță. Dintre aceste idei, două sunt remarcabile: exploatarea paralelismului procesoarelor (de obicei prin pipeline) și exploatarea accesului direct la memorie (de obicei prin completarea cu memorie de tip cache).

J. Presper Eckert și John Mauchly de la Școala Moore a Univ. Pennsylvania construiesc prima mașină cu caracteristicile constructive și funcționale ale unui calculator electronic operațional. Această mașină, numită **ENIAC (Electronic Numerical Integrator and Calculator)** a fost finanțată de SUA și folosită în timpul celui de-al doilea război mondial, dar nu a fost făcută publică până în 1946. ENIAC era folosită în special pentru calcularea tabelelor atacului de artilerie. ENIAC avea 80 picioare lungime, 8,5 picioare înălțime și câteva picioare adâncime. Fiecare dintre cei 20 regiștri de 10 biți avea 2 picioare lungime. În total, ENIAC folosea 18000 tuburi electronice. (1m = 3,2808 ft)

ENIAC executa salturi condiționate și era programabil. Programarea se făcea manual prin cuplarea (*plugging*) cablurilor și setarea comutatoarelor, iar datele erau introduse pe cartele perforate (*punched cards*). Programarea calculului de bază dura între o jumătate de oră și o zi întreagă.

În 1944 John von Neumann a fost atras în proiectul ENIAC. Grupul vroia să îmbunătățească modalitatea de introducere a datelor și se discuta asupra memorării programelor ca o succesiune de numere. Von Neumann a contribuit la cristalizarea ideilor și a scris un memo prin care propunea un calculator cu program memorat (program-stored computer), numit **EDVAC (Electronic Discrete Variable Automatic Computer)**. Acest memo este baza a ceea ce și astăzi numim calculator von Neumann (vezi Principiile von Neumann). Mulți apreciază că această denumire acordă prea mare credit lui von Neumann și neglijează aportul (inginerilor) creatorilor Eckert și Mauchly...

În 1946 Maurice Wilkes de la Universitatea Cambridge a vizitat Școala Moore. Întors la Cambridge, Wilkes s-a hotărât să demareze un proiect care să construiască un calculator cu program memorat, numit **EDSAC (Electronic Delay Storage Automatic Calculator)**. EDSAC a devenit funcțional în 1949 și se consideră primul calculator cu scopuri generale (engl., *general purpose calculator*), operațional și cu program memorat din lume.

În timpul celui de-al doilea război mondial au fost construite calculatoare specializate pentru decodificarea mesajelor interceptate de englezi de la nemți. O echipă de la Bletchley Park, din care făcea parte și Alan Turing, a construit în 1943 calculatorul Colossus. Această mașină a fost secretă până în 1970. După război, acest grup a avut o oarecare influență asupra pieței britanice de calculatoare.

În timp ce se lucra la ENIAC, Howard Aiken construia la Harvard un calculator electro-mecanic numit Mark-I. Acesta a fost urmat de Mark-II și apoi de două mașini cu tuburi electronice, Mark-III și Mark-IV. Față de EDSAC, care folosea o singură memorie atât pentru instrucțiuni, cât și pentru date, Mark-III și Mark-IV aveau memorii separate pentru instrucțiuni și pentru date. Aceste mașini au fost considerate împotriva calculatoarelor cu program memorat. Pentru a respecta importanța mașinilor lui Howard, astăzi, termenul de **arhitectură Harvard** descrie mașini cu o memorie principală unică, dar cu zone cache separate pentru date și instrucțiuni.

În 1947 a fost demarat la MIT (Massachusetts Institute of Technology) proiectul Whirlwind și avea ca scop realizarea unor aplicații pentru procesarea în timp real a semnalelor radar. Deși a condus la multe invenții, cea mai importantă rămâne memoria bazată pe miez (*core*) magnetic, care a fost baza sistemelor de memorare pentru următorii 30 de ani.

Firma IBM era implicată în afacerile cu cartele perforate și cu automatizarea sistemelor, dar nu a construit calculatoare până în anii 1950. Primul calculator IBM, **IBM 701**, a fost scos pe

piață în 1952 și a fost vândut în 19 exemplare. Imediat după 1950 mulți erau pesimiști în legătură cu viitorul calculatoarelor, mai ales gândindu-se că piața și oportunitatea pentru aceste mașini „super specializate” trebuie să fie foarte limitată.

În 1964, după o investiție de 5 miliarde de dolari, IBM a făcut un mare pas înainte când a anunțat calculatorul **System/360**. Acest sistem însemna o îmbunătățire a tuturor performanțelor de până atunci de cel puțin 25 de ori.

Un an mai târziu, DEC (Digital Equipment Corporation) a scos **PDP-8**, primul **minicalculator** comercial, la un preț sub 20.000 \$. Minicalculatoarele au fost precursorii miniprocesoarelor: în 1971 Intel a creat primul microprocesor, **Intel 4004**.

În 1963 s-a anunțat primul **supercalculator**, nu din partea companiilor mari sau a centrelor super-tehnologizate. Seymour Cray a creat în Minnesota mașina **CDC 6600** (CDC = Control Data Corporation). Această mașină includea multe din ideile regăsite în miniprocesoarele de mai târziu.

Cray va părăsi ulterior CDC pentru a forma Cray Research Inc. în Wisconsin. În 1976 Cray Research Inc. lansează Cray-1 care era în același timp și cea mai rapidă din lume, și cea mai scumpă, dar și calculatorul cu cel mai bun raport cost / performanță pentru programele științifice. În 1996 Cray Research este asimilată de Silicon Graphics.

În timp ce Cray crea cel mai scump calculator, alții se gândeau cum să folosească microprocesorul pentru a crea un calculator atât de „scump” cât să poată fi cumpărat pentru acasă. Au fost mai multe tentative de a lansa un **calculator personal**, dar remarcabil rămâne faptul că în 1977 Apple II (i.e. Steve Jobs și Steve Wozniak) a definit ce înseamnă: preț scăzut, volum mare și calitate bună pentru ceea ce avea să devină industria calculatoarelor personale.

Deși cu un avans de patru ani, Apple va sfârși pe locul al doilea... Lansat în 1981, **IBM Personal Computer** devine cel mai bine vândut calculator de orice tip. Succesul lui va da câștig de cauză lui Intel pentru piața de miniprocesoare și lui Microsoft pentru sistemul de operare. Chiar și astăzi, cel mai popular CD al lui Microsoft este cel cu sistemul de operare (Windows), chiar dacă este de multe ori mai scump decât un CD cu muzică!

Rezumat.

În acest paragraf am definit principalele concepte necesare pentru o bună înțelegere a structurii și funcționării calculatorului. Dintre acestea, se remarcă prin importanță definiția pentru **arhitectura calculatoarelor**, pentru că acest concept dă și titlul cursului.

Partea centrală a acestui prim paragraf din curs o constituie prezentarea structurii sistemului de calcul în diferite abordări: (1) tipurile de resurse ale calculatorului modern, (2) structura ierarhică de organizare a sistemului de calcul și (3) structura fizică și logică a sistemului de calcul.

Paragrafele următoare vor trata în detaliu tocmai componentele principale puse în evidență de structura logică a calculatorului, și anume: sistemul de intrare/ieșire (dispozitivele periferice) și unitatea de memorie, în capitolul întâi, structura și rolul regiștrilor și a UAL în capitolul al doilea, capitolul al treilea și al patrulea fiind rezervate structurii și funcționării unității centrale de prelucrare, concret a procesorului.

Cuvinte cheie.

sistem de calcul
hardware, software
arhitectura calculatoarelor, nivel conceptual de arhitectură
unitate funcțională, ecuație logică, UAL, UCP, memorie, IOS

Verificare.

1. Care sunt principalele dumneavoastră activități curente în care folosiți calculatorul?
2. Notați-vă principalele componente (denumire și caracteristici) ale calculatorului pe care îl folosiți în mod frecvent. Decideți care sunt elementele pe care le cunoașteți, pe care le

puteți descrie și care nu. Recitiți această descriere din când în când, pe măsură ce parcurgeți cursul și bifați elementele despre care tocmai ați învățat. Discutați în final cu cadrul didactic ceea ce a rămas nebifat pe lista dumneavoastră.

3. Care este legătura între structura fizică și structura logică ale sistemului de calcul?
4. Stabiliți-vă în scris câteva repere cronologice în evoluția sistemelor de calcul.

NOTIȚE

1.2. Sistemul de intrare/ieșire. Dispozitive periferice

Pentru a folosi SC, utilizatorul trebuie să introducă în calculator programele și datele aplicației sale. De asemenea, după rezolvarea tuturor sarcinilor primite, SC trebuie să înștiințeze utilizatorul asupra rezultatelor obținute.

Totalitatea unităților funcționale capabile să *organizeze și să memoreze informații externe*, pe de o parte și să *asigure schimbul de informații între SC și mediul extern*, pe de altă parte, alcătuiesc **sistemul de intrare/ieșire**. Concret, acesta cuprinde **dispozitivele periferice** sau, pe scurt, **perifericele**.

Principalele tipuri de dispozitive periferice legate la SC de tip IBM-PC sunt:

Consola (**tastatura**) – prin intermediul căreia utilizatorul introduce datele în calculator foarte comod, sub formă alfanumerică;

- **Mouse-ul** – un dispozitiv de manevrare a unui cursor grafic care permite folosirea comodă a interfeței grafice;
- **Imprimanta** – cu rol în transmiterea informațiilor din calculator pe hârtie. La SC recente, imprimanta dispune de diferite seturi de caractere și de puternice facilități grafice; *exemple*: imprimante cu ace, cu jet de cerneală, cu laser.
- **Hard-discul** – ca principal dispozitiv de memorare a informațiilor externe;
- **Unitatea de disc flexibil** – un SC poate avea una sau mai multe unități de disc flexibil. La început s-au folosit dischetele de 8". Au urmat cele de 5.25" și apoi unitățile de 3.5".
- **Unități de disc compact**, pentru citirea CD-urilor. Sistemele moderne au incorporate unități de disc compact *read-only*.

Pe lângă acestea, la SC se mai pot conecta și alte periferice, întâlnite mai rar în practică. Ele sunt destinate unor categorii de utilizatori specializați în anumite operații. Dintre acestea amintim: plotter, scanner, unitate ZIP, overhead, CD-writer.

Parametrii care caracterizează performanțele unui periferic sunt:

- ✓ modul de acces;
- ✓ timpul de acces;
- ✓ rata de transfer al informației;
- ✓ capacitatea;
- ✓ costul.

În funcție de tipul fiecărui dispozitiv periferic, unii parametri pot să nu aibă semnificație.

1.2.1 Clasificări

Din punctul de vedere al **direcției de transfer al informației**, avem: **periferice de intrare** (*input device*), **periferice de ieșire** (*output device*) și **periferice de intrare/ieșire** (*I/O device*). Din această ultimă categorie fac parte și rețelele interne destinate comunicării între periferice.

Din punct de vedere **funcțional**, dispozitivele periferice se împart în:

- 1) **periferice de schimb**: imprimanta, plotter-ul, videoterminalele, rețelele de comunicare cu alte periferice;
- 2) **periferice purtătoare de informații permanent pe medii magnetice**: benzile magnetice, discurile (hard-discurile, dischetele).

Preluarea datelor de pe aceste suporturi se face prin intermediul unui cap de citire. Benzile magnetice sunt dispozitive cu acces secvențial la informațiile memorate. Capul de citire are poziție fixă și el citește la un moment dat conținutul benzii care este în dreptul său.

La discurile magnetice, informația poate fi accesată în mod direct. Citirea se face prin deplasarea capului de citire până în dreptul zonei de pe disc unde este memorată informația căutată.

Observație. Discurile și benzile magnetice pot fi discutate și în contextul unităților de memorie auxiliară.

Din punctul de vedere al **tipului de transfer al informației**, avem: **periferice bloc** și **periferice caracter**.

Perifericele bloc sunt caracterizate de faptul că organizează informația în blocuri de lungime fixă, fiecare bloc având propria adresă.

Unitatea logică de schimb dintre perifericele *bazate pe suport magnetic* și memoria internă este **blocul de informație**. Acesta este format din unul sau mai multe sectoare vecine, care aparțin aceleiași piste. Un bloc va conține, pe lângă succesiunea propriu-zisă de octeți cu informație, un număr suplimentar de biți destinați verificării corectitudinii informației memorate în blocul respectiv. Pentru această completare sunt cunoscute două metode: schema de codificare polinomial-ciclică și schema cu biți de paritate încrucișată. *Exemple: mediile de tip disc.*

Perifericele caracter sunt caracterizate de faptul că furnizează sau primesc un *flux de octeți*, fără nici o structură de grupare a acestora. În consecință, octeții nu sunt adresabili și fiecare octet este disponibil ca și *caracter curent* până la apariția următorului caracter în/pe flux. *Exemple: imprimanta, terminalele cu tastatură și ecran, mouse-urile.*

Din punctul de vedere al **partenerului implicat în utilizarea perifericului respectiv**, avem: periferice care interacționează cu omul și periferice care interacționează cu mașina.

O caracteristică importantă pentru aprecierea unui periferic este dată de **rata de transfer**. Aceasta reprezintă numărul de unități de informație pe care le poate transfera perifericul respectiv într-o secundă în cadrul comunicării cu procesorul, cu memoriile sau cu alte periferice.

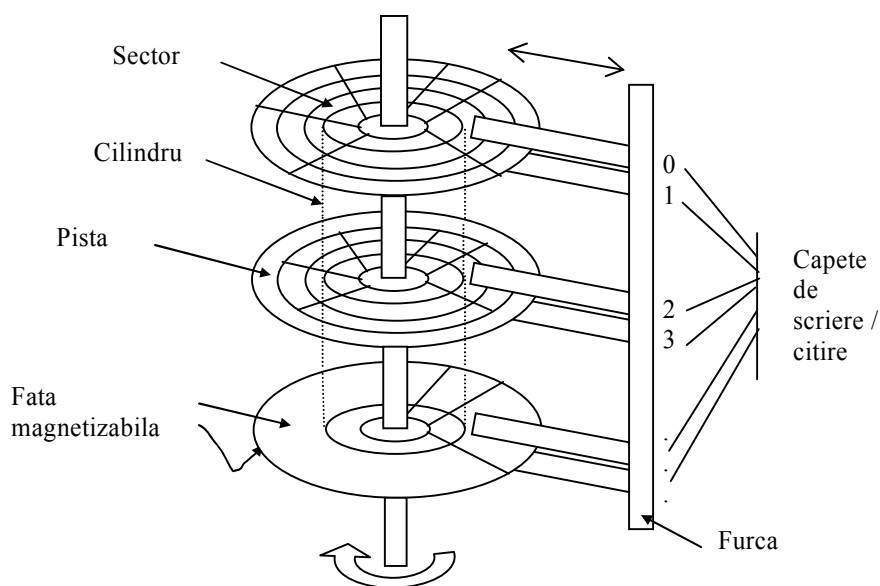
1.2.2 Caracterizarea unor periferice

1.2.2.1. Discurile dure. Hard-discul

Din punct de vedere fizic, un volum de disc magnetic are structura din figura următoare.

Volumul se montează pe o unitate de disc, unde se rotește cu o viteză constantă. Suprafața de memorare a unui disc este structurată pe trei nivele (pistă, cilindru, sector) și depinde de patru constante de construcție.

Suprafața fiecărei fețe active de memorare este divizată logic în coroane circulare concentrice numite **piste**. Numărul de piste de pe o față este prima constantă de construcție și ea variază între 30 și 800 piste/față.



În cazul în care volumul conține mai mult de două fețe active, situație frecvent întâlnită în practică, atunci fiecare față activă are același număr de piste și toate piste de aceeași rază

formează un **cilindru**. Numărul de fețe active ale unui volum de disc este a doua constantă de construcție și ea variază între 1 și zeci de fețe active/volum.

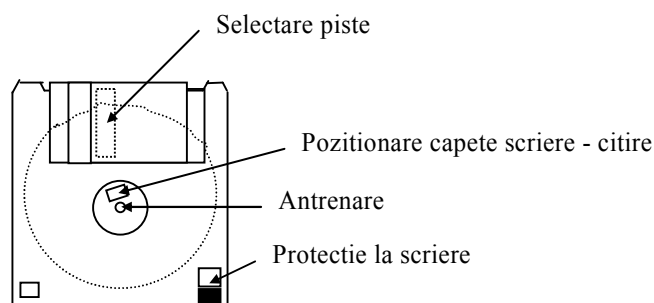
Fiecare pistă este împărțită în mai multe **sectoare**. Numărul de sectoare pe pistă este a treia constantă de construcție și ea variază între 4 și câteva zeci de sectoare/pistă. *Sectorul este unitatea de adresare a informației pe disc.*

În fine, numărul de octeți dintr-un sector este a patra constantă de construcție. De regulă, acest număr este o putere a lui 2 și variază între 128 și 4096 octeți/sector.

Hard-discul este creat după anul 1985 și rivalizează cu oricare dintre tipurile de discuri cunoscute. Capacitatea lor de memorare depășește capacitatea de memorare a altor discuri.

1.2.2.2. Discurile flexibile. Discheta

Un *disc flexibil* este format dintr-o folie magnetizabilă pe ambele fețe, îmbrăcată complet în material plastic nedeformabil (la discurile de 3.5"). Principal, îmbrăcăminte are prevăzute patru obturații (vezi figura următoare): una centrală pentru antrenare, una radială pentru selectarea pistelor și un orificiu în apropierea celui de antrenare, destinat poziționării capetelor de scriere - citire.



Al patrulea orificiu este în partea dreaptă jos: dischetele de 3.5" dispun (în vederea protecției la scriere) de un comutator cu două poziții: protejat la scriere, respectiv neprotejat. Astfel, dacă orificiul este închis atunci este permisă scrierea pe dischetă, iar dacă orificiul este deschis, se interzice scrierea.

Pentru o dischetă de 3.5" formatată la 1.44MB constantele de construcție sunt: 80 piste, 18 sectoare pe pistă, 2 capete de scriere-citire, 300 rotații pe minut, rată de transfer de 500Kbps (kilobiți pe secundă).

1.2.2.3. Compact discurile

Pentru gestionarea unor baze de date mari, casele de software utilizează din ce în ce mai mult discuri compacte, numite CD-ROM, ale căror capacități depășesc 1000 MO. În general, informațiile de pe CD-ROM-uri sunt de tip *read-only* și conțin de regulă baze mari de date cu caracter de consultare publică. Pe lângă CD-urile read-only, încep să se răspândească pe piață și CD-urile reinscriptibile.

Primele astfel de discuri conțineau baze de date documentare pe diverse domenii științifice: informatică, electronică, matematică, enciclopedii, etc.

1.2.2.4. Monitoare (graphic display)

Orice SC are un monitor (ecran) prin care comunică cu utilizatorii.

Funcționarea monitorului se bazează pe tehnologia de realizare a televizoarelor, adică este folosit un tub catodic (CRT - cathode ray tube). Astfel, un fascicul de raze scanează imaginea linie cu linie, de 30 - 75 ori pe secundă. La această rată de scanare utilizatorul nu poate observa raza pe ecran.

Imaginea poate fi interpretată ca o matrice de elemente luminoase elementare (*picture elements*), numite **pixeli**. De aceea, imaginea poate fi reprezentată printr-o matrice de biți, numită **bit-map**. Dimensiunile acestei matrice dau **rezoluția ecranului**. Cele mai simple

monitoare folosesc câte un bit pentru reprezentarea unui pixel și, astfel, pot reda numai imagini alb-negru. Urmează monitoarele care folosesc 8 biți pentru un pixel, care pot reprezenta astfel 256 nuanțe de gri ($256 = 2^8$). În fine, monitoarele color folosesc 24 biți pentru reprezentarea unui pixel și 8 biți pentru fiecare dintre culorile fundamentale, adică roșu, verde și albastru (RGB - Red, Green, Blue).

În particular, calculatoarele portabile au ecrane cu cristale lichide (LCD - liquid crystal display). Diferența esențială este faptul că un pixel LCD nu este o sursă de lumină.

Indiferent de tipul de monitor, componenta hardware pentru grafică conține mai multe buffere de imagine (*raster refresh buffer* sau *frame buffer*) pentru a memora imaginea sub formă de *bit map*. Imaginea de reprezentat pe ecran este memorată în *frame buffer* și, conform cu rata de actualizare a imaginii (*refresh rate*), fiecare bit al fiecărui pixel este citit, interpretat și reafășat pe ecran conform cu noile caracteristici (valori). Scopul realizării unui *bit map* este de a reprezenta fidel imaginea.

Competiția între sistemele grafice se dă pentru că ochiul uman detectează foarte bine fiecare schimbare de pe ecran. De exemplu, este neplăcut să se observe diferența dintre porțiunea de ecran pe care imaginea a fost actualizată, față de cea neactualizată.

Un monitor poate lucra în **regim text** sau în **regim grafic**.

Pentru conectarea monitorului la calculator se folosesc două modalități: legarea serială sau legarea prin adaptor video.

Ecranul poate lucra, dar nu simultan, fie în regim text, fie în regim grafic. În regim text vede (de regulă) 25 de linii a câte 80 de caractere fiecare, iar în modul grafic o matrice de pixeli, ale cărei dimensiuni depind de caracteristicile adaptorului video.

Evident că, pentru modul de lucru grafic memoria ecran este mult mai mare decât cea necesară pentru modul de lucru text.

În memoria ecran, fiecărui caracter (în mod de lucru text) sau fiecărui pixel (în mod de lucru grafic), îi sunt atașate așa numitele **attribute ale imaginii**. Aceste attribute se referă la:

- ✓ **strălucirea** punctului luminos (*brightness*);
- ✓ la posibilitatea imaginii de a **pulsa** pe ecran (*flashing*);
- ✓ la posibilitatea imaginii de a apărea în **video invers** (adică situația în care se schimbă culoarea fondului cu cea a punctelor luminoase);
- ✓ la **culoarea** pixelului sau a caracterului.

1.2.2.5. Mouse-ul

Prin "mouse" ne referim la perifericul de pointare (*pointing device*) inventat în 1967 și care a intrat în componența unui sistem de calcul de prin anii '80.

Principiul de funcționare se bazează pe utilizarea unei bile montată astfel încât să fie în contact cu doi cilindri de dimensiuni foarte mici. Unul dintre acești cilindri corespunde axei Ox, adică direcției orizontale, în timp ce celălalt corespunde axei Oy, adică direcției verticale.

Cilindrii sunt manevrați mecanic sau (la modelele moderne) antrenează câte o roțiță dințată prin intermediul căreia un LED (Light Emitting Diode) luminează un senzor foto.

Mișcarea fizică a mouse-ului pe *pad* determină învârtirea bilei, care determină rotirea cilindrilor Ox și Oy (după cum mouse-ul este mișcat orizontal, vertical sau pe diagonală) și, în continuare, actualizarea prin incrementare și/sau decrementare a unor contoare sistem. Aceste contoare (numărătoare) au rolul de a înregistra cât de departe a fost mișcat mouse-ul și în ce direcție.

Interfața între mouse și sistem este asigurată prin una dintre variantele:

- ✓ mișcarea mouse-ului generează impulsuri folosind LED-ul (vezi anterior) sau
- ✓ mișcarea mouse-ului generează operații de actualizare a valorilor unor contoare.

Periodic, procesorul semnalează impulsurile sau citește valorile contoarelor și determină deplasarea relativă a mouse-ului de la citirea anterioară. Conform cu rezultatul obținut, reprezentarea mouse-ului pe ecran este mutată proporțional. Mișcarea reprezentării mouse-ului

pe ecran este lentă deoarece variația poziției mouse-ului este mai mică decât rata de citire și interpretare a procesorului.

Mouse-ul este prevăzut cu două sau trei butoane. Operația esențială în lucrul cu mouse-ul este cea de **eliberare** a unuia dintre butoane.

Legătura dintre valorile contoarelor, poziția (starea) butoanelor și poziția reprezentării mouse-ului pe ecran se face prin mecanisme software. De obicei, aceste programe sunt grupate în *driver*-ul de mouse și se cumpără împreună cu perifericul. Deoarece există acest soft, utilizatorul poate seta diverși parametri de lucru cu mouse-ul: intervalul de timp caracteristic unei operații *double-click*, viteza de deplasare a reprezentării mouse-ului pe ecran, s.a.

Pe de altă parte, interpretarea permanentă a poziției mouse-ului pe ecran asigură că niciodată mouse-ul „nu va sări din ecran”. Metoda prin care sistemul primește informații despre mouse prin citirea și interpretarea semnalelor de la mouse se numește *polling* [poulin].

1.2.2.6. Rețele de comunicare

Dacă luăm în discuție comunicarea prin rețele atunci distingem două tipuri de rețele de comunicare: rețeaua internă a sistemului de calcul și rețelele de comunicare între sisteme de calcul.

Rețeaua internă a unui SC interconectează componentele fizice interne ale sistemului, concret, procesorul la memorii și la dispozitivele periferice.

Rețelele de comunicare între mai multe SC rezultă în urma interconectării componentelor a diferite calculatoare (sisteme multicalculator). Principalele avantaje ale realizării și utilizării unei rețele de calculatoare sunt: comunicarea rapidă și facilă, posibilitatea partajării resurselor, accesul la distanță.

1.2.2.7. Placa de rețea

Placa de rețea este componenta fizică a unui sistem de calcul prin care acesta se poate conecta la o rețea de calculatoare. O placă de rețea funcționează ca interfață fizică între calculator și cablul de rețea. Fiecare calculator din rețea (stație) și server-ul rețelei vor avea instalată câte o placă de rețea (fizic, într-un slot pe placa de bază sau incorporată direct în placa de bază). Legătura fizică între calculator și restul rețelei se stabilește după ce cablul de rețea se conectează la portul plăcii de rețea.

Fiecare „participare” a calculatorului în rețea presupune un transfer de date între calculator și placa de rețea, deci un proces de comunicare. În cazul plăcilor cu acces direct la memorie, **DMA Board** (**D**irect **M**emory **A**ccess), calculatorul-stație alocă o parte din spațiul său de memorie pentru placa de rețea.

1.2.2.8. Modemul

Atunci când se pune problema ca două calculatoare să comunice prin intermediul unei linii telefonice este nevoie și de un dispozitiv numit **modem**. Necesitatea modemului rezultă din faptul că liniile telefonice pot transporta numai semnale analogice (sunet), în timp ce calculatoarele comunică prin impulsuri digitale (semnale electronice). Un semnal digital este unul discret, care are una din două valori posibile: 0 sau 1. Un semnal analogic poate fi reprezentat printr-o curbă continuă, având un domeniu infinit de valori.

Astfel, modemul apare ca un **modulator-demodulator** de semnale, care convertește semnalele digitale în semnale analogice și invers.

Un modem include o interfață de comunicare serială (RS-232, vezi anterior *Legarea serială*) și o interfață pentru linia telefonică, RJ-11 (adică o priză de telefon cu patru fire).

Modemurile sunt disponibile atât ca modele interne, cât și externe. Un **modem intern** este instalat într-un slot de extensie al calculatorului, la fel ca orice altă placă de interfață. Un **modem extern** este ca o cutie conectată la calculator prin intermediul unui cablu serial (RS-232), care face legătura între portul serial al calculatorului și conectorul de interfață serială al

modemului. În plus, indiferent de model, modemul folosește și un cablu cu conector RJ-11C pentru a se conecta la priza telefonică de perete.

1.2.3 Legarea perifericelor la SC. Unitățile de interfață I/O

Perifericele conectate la un SC au nevoie de linii speciale de comunicare pentru a se lega la UCP. Concret, aceasta se realizează prin componente hardware numite **unități de interfață** cu rol în supervizarea și sincronizarea tuturor transferurilor de intrare / ieșire.

Principalele funcții ale unităților de interfață sunt:

- ◆ conversia semnalelor specifice modului de operare pe suporturi magnetice în semnale de lucru pe circuite electronice, adică de pe periferic magnetic în CPU;
- ◆ sincronizarea ratelor de transfer de date de la viteza de lucru pe periferic la viteza de lucru în CPU;
- ◆ decodificarea/codificarea datelor de pe periferic în format specific în CPU;
- ◆ delimitarea semnalelor unui periferic de ale altora dintre cele conectate la același SC.

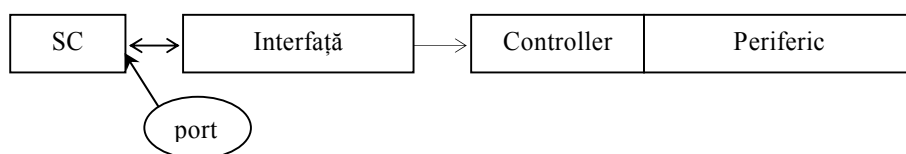
În funcție de numărul semnalelor de date prin care se realizează transmiterea informației dinspre periferic distingem **interfețe seriale** și **interfețe paralele**.

Comunicarea serială presupune că informația circulă într-o structură secvențială de biți în care alternează biții de informație cu biții de control. Numărul și semnificația biților de control sunt conform cu protocolul de comunicare prin care se asigură transmisia datelor. În general, avem un bit de start, unul sau doi biți de stop și un bit de paritate. În comunicarea paralelă, informația circulă în blocuri de 8 biți. Avantajul comunicării paralele constă în mărirea vitezei de transfer și în eliminarea biților de control.

În plus, comunicarea, respectiv transmisia datelor, poate fi cu sincronizare (sau sincronă) dacă semnalele de date sunt completate de semnale de tact (sau de ceas). În acest caz, viteza de transmisie a datelor depinde de frecvența de tact. Dacă nu există semnale de tact atunci comunicarea este asincronă și sincronizarea se poate realiza prin însăși structura datelor transmise, de exemplu, comunicarea bazată pe caracter sau comunicarea bazată pe mesaj.

Pe lângă interfață, fiecare periferic poate avea o unitate de control proprie, numită **controller**, cu rol în supervizarea operațiilor specifice mecanismului de funcționare a perifericului respectiv. De exemplu, controllerul propriu imprimantei gestionează deplasarea hârtiei, timpul de listare și selectarea tipului de caracter de imprimat.

Controller-ul poate fi independent sau poate fi integrat fizic cu perifericul.



Punctul în care se conectează fizic interfața la SC este un registru de date care se numește **port**. Practic, portul este registrul prin care se realizează schimbul de informații între sistem și exterior. În funcție de tipul de interfață, acesta poate fi **port de intrare**, **port de ieșire** sau **port de intrare / ieșire**.

Portul de intrare / ieșire specifică un canal prin care informația circulă între dispozitivele hardware și UCP. Portul este identificat de UCP printr-o adresă. Astfel, fiecare resursă hardware din sistem trebuie să aibă o altă adresă de port I/O.

Concret, comunicarea între procesor și diferite periferice se face prin **magistrala de intrare / ieșire** (I/O Bus). Cuplarea perifericelor la sistemul de calcul presupune, pe de o parte, adaptarea semnalelor specifice fiecărui echipament la semnalele de pe magistrală și, pe de altă parte, reglarea fluxului de date între calculator și periferice.

Din punct de vedere logic, legătura între dispozitivele periferice, memorii și UCP este asigurată de o rutină de interfață care se va numi **driver**. Concret, un driver este o aplicație software care identifică perifericul în sistem.

Comunicarea între un periferic și UCP este permanentă. În acest sens, procesorul citește și interpretează periodic anumite date despre starea fiecărui periferic. Subsistemul de gestiune a perifericelor poate testa periodic fiecare periferic la același interval de timp sau pe baza unui sistem de priorități. Tehnica „sondării” periodice de către procesor a stării perifericelor se numește **polling** [poulin]. Testarea pe baza unui sistem de priorități este o metodă „de încredere” și poate fi cea mai potrivită soluție în sistemele mici. Tehnica *polling* este o metodă nerecomandată în sistemele puternic dependente de timp (*time-critical*) deoarece un eveniment important poate apărea imediat ce perifericul său tocmai a fost testat; în acest caz evenimentul este lăsat în așteptare până la următoarea testare, ceea ce nu este convenabil pentru un eveniment important. O alternativă modernă, deosebit de utilă în toate tipurile de sisteme, este folosirea întreruperilor.

1.2.4 Magistrale

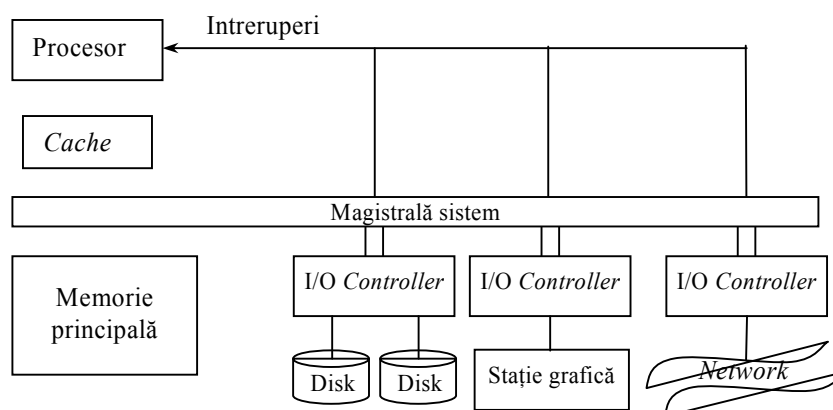
Conform cu arhitectura von Neumann a sistemelor de calcul comunicarea între componente se realizează prin legături dedicate între perechi de componente. Aceasta este o abordare rigidă, care limitează scalabilitatea sistemului. Alternativa pentru această situație este soluția propusă de firma DEC (*Digital Equipment Corporation*) care, la sfârșitul anilor '60, a lansat pe piață primul calculator (și naume PDP 11) construit în jurul unei magistrale unice, magistrala UniBus.

Conceptual, o **magistrală** este un mediu comun de comunicare între componentele sistemului de calcul. Din punct de vedere fizic, o **magistrală** este un set de linii de semnal (date, adrese, comenzi, control, întreruperi, tact) care facilitează transferul de date și sincronizarea componentelor conectate la magistrala respectivă.

Caracterul de sistem deschis al calculatoarelor moderne este susținut în mare măsură de standardizarea magistrelor. Astfel, avem astăzi două clase importante de magistrale: (1) *magistrale de sistem* – dezvoltate pentru conectarea UC (unității centrale) la celelalte componente ale calculatorului; exemple: MultiBus, ISA, EISA, PCI și USB; (2) *magistrale specializate* – dezvoltate pentru a optimiza transferul de date către/dinspre un anumit periferic; exemple: VESA, SCSI, GPIB. În ultima perioadă se observă o extindere a standardelor de magistrală pentru a răspunde cerințelor de comunicare ale noilor generații de procesoare.

Ținând cont de componentele sistemelor de calcul introduse în paragrafele anterioare, putem spune că o magistrală conectează dispozitivele de intrare - ieșire la procesor și la memorii. Mai mult, o magistrală reprezintă calea de comunicare partajată care folosește un set de conductori pentru a inter-conecta subsistemele respective. Magistrala reprezintă interconectarea electrică între periferice, procesoare și memorii și, deci, definește protocolul de comunicare la nivelul cel mai de jos (*lowest level*).

Un ansamblu tipic de periferice conectate la procesor și memorii se poate reprezenta ca în figura următoare.



Folosirea magistralelor de comunicare are în principal avantaje legate de flexibilitatea conectării componentelor. Astfel, pentru o schemă deja definită se pot adăuga cu ușurință noi tipuri de periferice și, mai mult, perifericele pot fi mutate de la un SC la altul, dacă acestea folosesc același tip de magistrală.

În cazul în care magistrala este suprasolicitată, apare dezavantajul major al comunicării „gâtuite” (engl., *bottlenecked*). Această situație poate fi evitată prin fixarea (și respectarea) unui număr maxim de intrări (periferice conectate) pe magistrală.

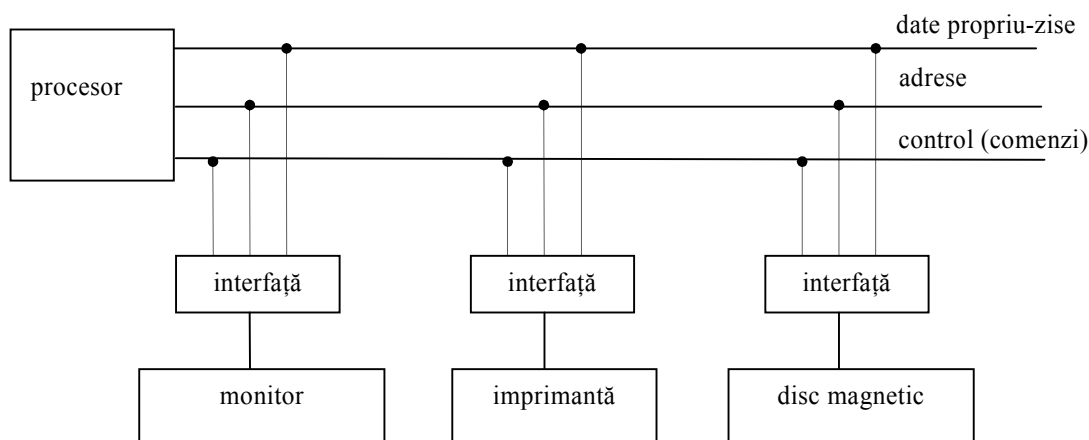
Performanțele unei magistrale se apreciază din două puncte de vedere fundamentale: viteza maximă de transfer a datelor pe acea magistrală și numărul maxim de periferice pe care le poate conecta. Scopul concret pentru care este construit fiecare model de magistrală va decide care dintre aceste criterii este prioritar și, în consecință, va valorifica la maximum acea caracteristică, în detrimentul celeilalte. Proiectarea unui model este dificilă, ținând cont și de faptul că viteza de transfer este limitată la rândul ei de factori fizici cum ar fi: numărul de periferice conectate și lungimea magistralei.

Conform tipului de informații care circulă pe magistrală avem: magistrală de date, magistrală de adrese și magistrală de control (comenzi). Generic, ansamblul tuturor acestor linii este recunoscut ca magistrală sistem. Numărul liniilor de un anumit tip este proporțional cu viteza transferului pe liniile respective.

Pe liniile de date se transferă informații de la o entitate sursă la o entitate destinație. Aceste informații pot fi: date, comenzi sau adrese. Rolul liniilor de control este: să semnaleze cererile, să precizeze tipul datelor de pe liniile de date, să implementeze protocolul de magistrală. Deoarece magistrala este partajată, trebuie stabilit un protocol care să decidă univoc la fiecare moment cine folosește magistrala la momentul următor. Operația de bază cu care lucrează un protocol de magistrală este **tranzacția pe magistrală**, care constă din trimiterea adresei, pe de o parte și transmiterea și primirea datelor, pe de altă parte. O astfel de tranzacție este definită de direcția de lucru cu memoria, rezultând **tranzacții de intrare** și **tranzacții de ieșire**.

Exemplu de utilizare a unei magistrale sistem. Dacă un disc vrea să scrie date în memorie de pe un anumit sector al său atunci liniile de date se vor folosi pentru a preciza adresa din memorie la care se va face scrierea datelor și, ulterior, tot pe liniile de date se vor transfera și datele propriu-zise de pe disc. În acest caz, liniile de control vor indica ce tip de date (adresă sau date propriu-zise) sunt pe liniile de date la fiecare moment al tranzacției.

Observație. Unele magistrale au două seturi de linii de date pentru a separa datele de adrese. În continuare este descris un model de transfer pe o magistrală cu linii de date, de adresă și de control.



Liniile de magistrală I/O care pornesc de la procesor sunt legate la unitățile de interfață ale tuturor perifericelor. Pentru a comunica cu un anumit periferic, procesorul pune adresa perifericului pe liniile de adresă. Fiecare interfață conectată la magistrala I/O conține un

decodificator de adrese care gestionează liniile de adresă. Când o interfață detectează propria sa adresă, deschide accesul între liniile de magistrală și perifericul pe care îl gestionează. Pentru toate interfețele cărora nu le corespunde adresa de pe magistrală, perifericele respective sunt inaccesibile.

Când procesorul pune adresa perifericului căutat pe liniile de adresă, în același timp furnizează pe liniile de control și codul unei funcții (*function code*). Interfața selectată răspunde acestui cod și determină execuția funcției. Codul acesta este o **comandă de intrare/ieșire** și constă într-o instrucțiune de executat de către interfață și perifericul asociat ei. Interpretarea comenzii depinde de perifericul căreia îi este adresată. O interfață de periferic poate primi următoarele patru tipuri de comenzi:

- ✓ comenzi de control – care activează perifericul sau îl informează asupra următoarei operații;
- ✓ comenzi de stare – cu rol în testarea stării interfeței și a perifericului. Erorile care apar la transferul datelor sunt semnalate prin valorile anumitor biți din registrul de stare. Procesorul citește periodic acest registru, detectând astfel situația în care a apărut o eroare;
- ✓ comenzi de ieșire – care constau în comenzi de transfer al datelor înspre exterior, adică determină interfața să răspundă prin dirijare de date de pe magistrală într-unul din regiștrii săi de tranzit (*buffer register*);
- ✓ comenzi de intrare – care sunt opuse comenzilor de ieșire. În acest caz, interfața primește un set de date de la periferic, pe care le depune într-unul din regiștrii săi de tranzit. Procesorul verifică printr-o comandă de stare dacă datele sunt accesibile și apoi transmite o comandă de intrare. Interfața depune datele pe liniile de date ale magistralei I/O de unde sunt preluate de procesor.

1.2.4.1. Clasificarea magistralelor

Din punctul de vedere al componentelor conectate, avem:

- ✓ magistrale procesor - memorie;
- ✓ magistrale de intrare - ieșire;
- ✓ magistrale de extensie (*backplane*).

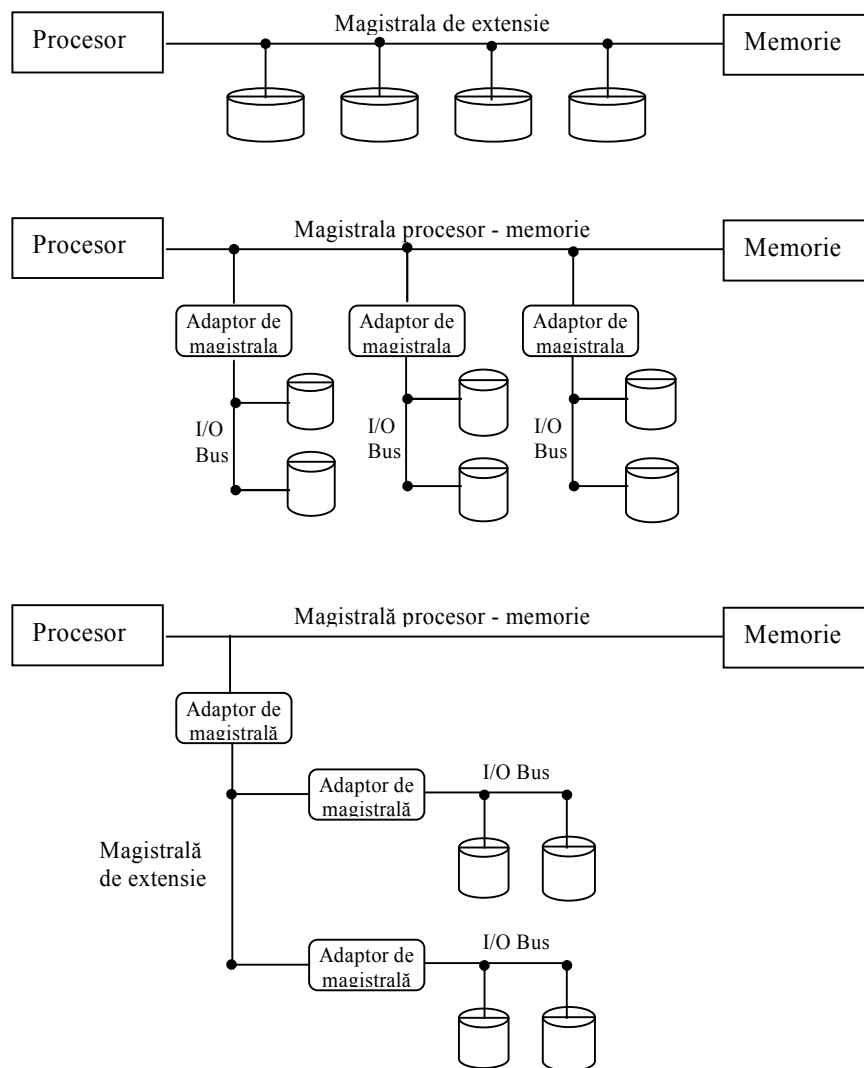
Magistralele procesor - memorie sunt, în general, scurte, dar de viteză mare și adaptate sistemului de memorie, astfel încât să maximizeze lungimea de bandă procesor - memorie și, implicit, cantitatea de informație transmisă simultan.

Magistralele de intrare - ieșire (I/O Bus) pot fi suficient de lungi, pot conecta mai multe tipuri de periferice și acceptă o varietate mare de lungimi de bandă pentru perifericele conectate. Magistralele de intrare-ieșire nu interacționează direct cu memoria ci se folosește o magistrală procesor - memorie sau una de extensie pentru conexiunea cu memoria.

Magistralele de extensie (*backplane*) sunt construite pentru ca procesoarele, memoriile și perifericele să poată coexista pe aceeași magistrală. Astfel, o magistrală de extensie va răspunde atât cererilor de comunicare procesor - memorie, cât și cererilor de comunicare dintre periferice și memorie. Se numesc de extensie tocmai pentru că permit adăugarea în sistem a diferitelor plăci interschimbabile, prin inserarea acestor plăci în conectorii de extensie ai magistralei (sloturi).

Comparativ, putem spune că magistralele procesor - memorie au design specific și sunt, în general, monopol al firmei producătoare, în timp ce celelalte două tipuri (de intrare - ieșire și de extensie) pot fi refolosite în diferite sisteme și sunt construite pe baza unor **standarde de magistrală**.

În figurile următoare reprezentăm trei variante de interconectare a componentelor SC prin magistrale de comunicare.



Avantajele remarcabile ale variantei a treia, cea mai complexă, ar fi:

1. magistrala procesor - memorie poate fi mult mai rapidă decât magistralele de extensie sau de intrare - ieșire;
2. sistemul de intrare - ieșire poate fi completat cu multe *controller*-e sau chiar cu magistrale I/O pe magistrala de extensie — ceea ce nu afectează viteza magistralei procesor - memorie.

Din punctul de vedere al tipului de comunicare, avem:

- ✓ magistrale sincrone;
- ✓ magistrale asincrone.

O **magistrală sincronă** are pe liniile de control un ceas (generator de tact) care controlează un protocol bine definit pentru comunicare. De obicei, magistralele procesor - memorie sunt magistrale sincrone, deoarece conectează puține componente și trebuie să lucreze la viteză mare de transfer a datelor.

Observație. Funcționarea unei magistrale sincrone poate fi simulată cu un automat finit determinist.

Prezența ceasului impune restricții în construirea și funcționarea unei magistrale sincrone. De exemplu, toate componentele conectate trebuie să funcționeze la aceeași frecvență de tact. De asemenea, datorită problemelor de sincronizare, o magistrală sincronă nu poate fi și lungă și rapidă.

O **magistrală asincronă** nu este controlată de ceas, deci poate grupa o mai mare varietate de periferice și poate fi oricât de lungă. În acest caz, pentru coordonarea transmiterii de date se

folosește un protocol de tip *hand shaking* (vezi și Moris Manno, pag. 393). Acesta constă dintr-o succesiune de pași în care sursa și destinatarul transferului de date trec la pasul următor numai de comun acord. Acest protocol este practic implementat cu suplimentarea liniilor de control pe magistrală.

Observație. Funcționarea unei magistrale asincrone poate fi simulată cu o pereche de automate finit deterministe: o mașină trece în starea următoare numai dacă cealaltă a intrat într-o anumită stare - test.

1.2.4.2. Standarde de magistrală

Magistralele de intrare - ieșire servesc ca o modalitate de a îmbunătăți performanțele SC prin conectarea a noi periferice. Pentru a facilita această operație, industria sistemelor de calcul a dezvoltat o serie de standarde de magistrală.

Un **standard** reprezintă o specificare pentru un producător de componentă. Astfel, un **standard de magistrală** impune producătorilor structura de bază a magistralei, astfel încât să poată fi cunoscută și folosită a priori de producătorii perifericelor, cu compatibilitate maximă cu mașina gazdă.

Un standard uzual de magistrală este, de exemplu, **SCSI** – *Small Computer System Interface*. De obicei, o magistrală de acest tip este o interfață fizică către o magistrală de extensie sau către o magistrală procesor - memorie. Un *controller* SCSI coordonează transferurile de la periferic, pe magistrala de intrare - ieșire, spre memorie, prin intermediul unei magistrale procesor - memorie. Deci, magistralele de intrare - ieșire sunt construite conform acestui standard.

Pentru magistralele de extensie este recunoscut standardul **ISA** – *Industrial Standard Architecture*. Inițial au fost magistrale pe 8 biți și, ulterior, au fost extinse la 16 biți, ca standard pentru IBM PC/AT. Limitată la o frecvență de ceas de 8 MHz, această magistrală a fost deja insuficientă pentru procesoarele 80386. Totuși, o magistrală ISA (sloturi de 8 biți sau de 16 biți) rămâne o opțiune pentru conectarea imprimantelor prin porturi seriale, pentru *controller*-e de mouse, dar sunt prea lente pentru discurile moderne sau pentru adaptoare video.

Standardul **PCI** – *Peripheral Component Interconnect* – a fost inițiat de Intel și dezvoltat ulterior de alt concern industrial. Aceste magistrale sunt cele de mare viteză. PCI a fost prima magistrală care a răspuns cererilor de magistrală pe 32 biți reclamate de procesoarele Pentium și opera la frecvențe de tact de 33 MHz sau 66 MHz, permițând rate de transfer de 132 MB / sec sau 264 MB / sec. PCI suportă controlul total al magistralei (*bus mastering*). De obicei, magistralele de extensie sunt magistrale PCI. Arhitectura magistralei PCI poate oferi facilitatea *Plug-and-Play* (autoconfigurare). Această tehnologie constă în adaptarea automată a configurației calculatorului (fără intervenția utilizatorului) la caracteristicile componentei care tocmai se instalează. Astfel, instalarea oricărui dispozitiv devine o operație simplă și sigură. În plus, autoconfigurarea face inutilă setarea manuală a configurației calculatorului. Sistemele de operare Windows moderne recunosc facilitatea *Plug-and-Play*.

Calculatoarele personale dispun de patru tipuri de arhitecturi de magistrală și anume: ISA, EISA (Extended ISA), Micro Channel și PCI. Magistrala PCI este adecvată pentru atașarea la calculator a perifericelor de viteză mare, dar este prea costisitor să existe câte o interfață PCI pentru fiecare periferic de viteză scăzută, de exemplu, tastatura, *mouse*-ul, camera foto/video digitală, scanner, telefon digital ș.a.m.d.. De aceea, pentru conectarea la calculator a dispozitivelor periferice de viteză redusă s-a impus un standard de magistrală unanim acceptat, anume **USB** (engl. *Universal Serial Bus*).

Rezumat.

Scopul acestui paragraf a fost prezentarea caracteristicilor esențiale ale principalelor periferice conectate la un calculator modern, adică periferice standard (tastatură, mouse, monitor) și periferice larg utilizate (discuri, imprimantă, modem, plăci de conexiune). Tratarea

acestor periferice s-a făcut din mai multe puncte de vedere: (1) identificarea diferitelor tipuri de periferice (clasificări), (2) caracterizarea elementelor specifice fiecărui tip de periferic, (3) caracterizarea perifericelor din perspectiva conectării lor cu componentele cu care comunică direct.

Cuvinte cheie.

dispozitiv periferic, sistemul de intrare/ieșire
periferic de intrare, periferic de ieșire, periferic de intrare/ieșire
periferic de schimb, periferic de memorare
unitate de interfață, port
magistrală, standard de magistrală

Verificare.

1. Care sunt perifericele conectate la sistemul de calcul pe care îl folosiți în mod frecvent?
2. Explicați modul în care procesorul central stabilește comunicarea directă cu un anumit periferic solicitat la un moment dat.

NOTIȚE

1.3. Unitatea de memorie

Unitatea de memorie este o colecție de componente destinate memorării datelor și de circuite asociate necesare transferului de informații înspre și dinspre memorie.

În general, componentele de memorie au rolul de a stoca informația utilizată de unitatea centrală și de dispozitivele periferice.

1.3.1 Structura fizică a memoriei

Referitor la *structura fizică* a memoriei, avem că **memoria** este formată dintr-un mediu de memorare și sisteme de circuite electromagnetice pentru controlul gestiunii memoriei. La rândul său, din 1919 subsistemul de memorare este alcătuit dintr-o succesiune de **bistabili** (circuite basculante bistabile) ce permit sau nu trecerea curentului, fiecare stabilind astfel memorarea uneia dintre valorile 0 sau 1.

Observație. Reprezentarea a două stări posibile ale unui sistem prin 0 și 1 a determinat folosirea în sistemele de calcul în special a bazei de numerație 2 și a puterilor lui 2.

În general, cantitatea de informație care se câștigă prin precizarea stării unui sistem care are două stări posibile se numește **bit de informație**. În particular pentru memorii, vom înțelege prin **bit** informația referitoare la starea unui bistabil: 0 sau 1. Cu alte cuvinte, un bistabil îndeplinește funcția de memorare a unui bit de informație.

În memorie, un grup de bistabili se numește **locăție de memorie**. Pentru SC actuale, o locăție are opt biți și se numește **octet (O)**, sau **byte (B)**, sau **caracter**. Această echivalență de denumire provine de la faptul că un caracter se reprezintă în memorie pe un octet.

Octeții sunt numerotați de la 0 la capacitatea maximă a memoriei. Astfel, capacitatea memoriei este dată de cantitatea de informație pe care aceasta o poate stoca, adică numărul de octeți dați eventual, multiplicativ: 1 KB (kilobyte) = 210B = 1024 octeți, 1 MB (megabyte) = 210 KB = 220 octeți, 1GB (gigabyte) = 210 MB = 230 octeți ($230 = (210)^3 \approx (103)^3 = 109$), 1TB (terabyte) = 210 GB = 240 octeți. Deoarece reprezentarea internă a datelor în memoria sistemelor de calcul este binară și nu zecimală, producătorii de calculatoare și-au însușit aceste estimări binare ale prefixelor din Sistemul Internațional de Unități [Wikipedia, *Since computer memory comes in multiples of 2 rather than 10, the industry used binary estimates of the SI-prefixed quantities*. <http://en.wikipedia.org/wiki/Byte>].

Numărul octetului reprezintă **adresa de memorie** a grupului respectiv. Această adresă joacă un rol esențial în regăsirea informației memorate și transmiterea ei către alte subsisteme. **De exemplu**, dacă unitatea centrală cere transmiterea unei informații memorate atunci ea trebuie să furnizeze atât *adresa octetului de unde se face transferul*, cât și *numărul de octeți care se transferă*.

În terminologia modernă, numim **cuvânt** orice succesiune de 2, 4 sau 8 octeți. Astfel, calculatoarele din primele generații, până la 80286 inclusiv, au cuvântul format din 2 octeți, începând cu seria 80386, cuvântul este de 4 octeți, iar la microcalculatoarele moderne avem cuvântul de 32 biți = 4 octeți sau 64 biți = 8 octeți. Supercalculatoarele au cuvântul format din 8 octeți. Lungimea cuvântului este o caracteristică a procesorului.

Biții unui cuvânt trebuiesc ordonați într-un mod unic. Pentru aceasta se folosește ordinea puterilor lui 2. Toate tipurile de SC numerotează biții începând cu 0. Dintre biții unui octet se remarcă:

- **bitul cel mai semnificativ, MSB**, cel care corespunde puterii celei mai mari a lui 2;
- **bitul cel mai puțin semnificativ, LSB**, cel care corespunde puterii celei mai mici a lui 2.

În funcție de modelul de procesor, biții unui octet sunt numerotați crescător de la stânga la dreapta (format *big endian*, specific procesoarelor Motorola) sau de la dreapta la stânga (format *little endian*, specific procesoarelor Intel).

1.3.2 Tehnologii de realizare a memoriilor. Memorii semiconductoare

Cronologic, principalele tehnologii de realizare a componentelor de memorare sunt: (1) tambur magnetic, (2) inele de ferită, (3) discuri magnetice, (4) memorii semiconductoare, (5) procedee optice.

Memoriile semiconductoare sunt circuite integrate bazate pe metale semiconductoare, de obicei Siliciu. Față de Germaniu, Siliciul este preferat deoarece își păstrează conductibilitatea chiar și la temperaturi înalte.

În sistemele de calcul binare, circuitele de memorie stochează informația binară în celule elementare de memorie cu capacitatea de 1 bit. Celulele de memorie sunt grupate în locații de memorie, care, în funcție de tipul de memorie, poate fi de 1, 2, 4 sau 8 celule de bit. Impactul comercial pe care l-au avut procesoarele pe 8 biți în anii '80 a impus dimensiunea uzuală de 8 biți pentru o locație de memorie (un octet sau un *byte*). Fiecărei astfel de locații îi este asociat un număr unic recunoscut ca **adresa** locației respective. Combinația binară citită / înscrisă într-o locație de memorie reprezintă conținutul locației respective și este un **cuvânt-memorie**. Numărul total de locații dintr-o componentă de memorare definește capacitatea circuitului de memorie respectiv. Intervalul (domeniul) adreselor definește **spațiul de adresare** al circuitului de memorie.

O memorie semiconductoare este caracterizată de următorii parametri:

- ✓ **geometria** (sau modul de organizare a memoriei) – se referă la: modul de dispunere a celulelor bit, lungimea unui cuvânt-memorie, modul de adresare a cuvintelor;
- ✓ **capacitatea** – exprimată în numărul de cuvinte – memorie; din punct de vedere practic, capacitatea unei memorii este dată de cantitatea maximă de informație care poate fi stocată la un moment dat pe memoria respectivă, măsurată în *bytes*;
- ✓ **timpul de acces** – reprezintă diferența dintre momentul în care memoria primește (pe liniile de intrare în circuit) adresa locației solicitate și momentul în care memoria furnizează (pe liniile de ieșire din circuit) datele conținute la adresa respectivă ($t_{\text{acces}} = t_{\text{DataOut}} - t_{\text{AdrIn}}$); se măsoară în *ns* sau μs ;
- ✓ **ciclul memoriei** – reprezintă timpul necesar pentru citirea/scrierea conținutului unei locații; cu alte cuvinte, ciclul memoriei este intervalul de timp dintre două aplicări succesive de adrese pe liniile de intrare în circuitul de memorie; se măsoară în *ns* sau μs ;
- ✓ **puterea consumată** – se măsoară în $\mu W / bit$;
- ✓ **volatilitatea** – o memorie este volatilă dacă își pierde conținutul în timp (memorii RAM dinamice, DRAM) sau la decuplarea de la sursa de alimentare electrică (memorii RAM statice, SRAM); memoriile de tip ROM sunt nevolatile;
- ✓ **tehnologia de realizare** – din acest punct de vedere avem:
 - *memorii bipolare*: foarte rapide, dar cu densitate mică de integrare; principalele tehnologii bipolare sunt: TTL (*Transistor-Transistor Logic*) și ECL (*Emitter-Coupled Logic*);
 - *memorii MOS (Metal-Oxide Semiconductor)*: cu densitate mare de integrare, putere consumată mică, dar cu timp mai mare de acces decât memoriile bipolare; memoriile de tip flash (*USB flash drive*) sunt realizate cu tehnologii de tip MOS.

Observație. Din definițiile de mai sus se observă că timpul de acces la memorie și ciclul memoriei caracterizează implicit și performanțele operaționale ale unei memorii (concret, viteza de execuție a operațiilor pe circuitul de memorie respectiv).

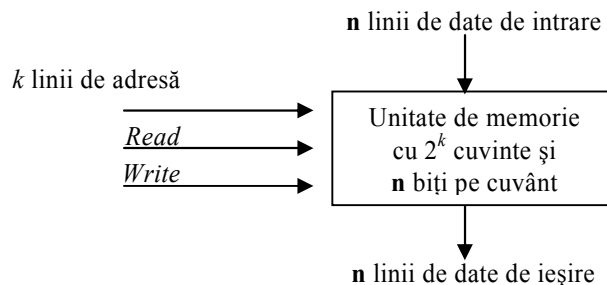
Principalele operații efectuate de circuitele de memorii semiconductoare și modelele corespunzătoare de memorii sunt:

- ✓ **citire** – memorii ROM (*Read Only Memory*);
- ✓ **citire și scriere** – memorii RAM (*Random Access Memory*); în particular, avem:
 - *memorii RAM statice* – mențin informația cât timp este menținută tensiunea de alimentare; uzual, celula de bit a unei memorii RAM statice este un bistabil;

- *memorii RAM dinamice* – informația se pierde în timp, chiar dacă tensiunea de alimentare nu se întrerupe; uzual, celula de bit a unei memorii RAM dinamice este un condensator (nu un bistabil).
- ✓ **programare** – memorii PROM (*Programmable Read Only Memory*);
- ✓ **regenerare** – operație necesară, de exemplu în cazul memoriilor RAM dinamice; regenerarea este o citire incompletă (i.e. datele citite nu se utilizează în exteriorul circuitului ci, periodic, conținutul celulei de bit este selectat, citit și rescris intern în circuitul de memorie); uzual, rata de regenerare este de $\sim 2\text{ms}$ și nu depinde de capacitate.

Memoriile RAM sunt cu *acces direct*, deci informațiile pot fi accesate pentru transfer din orice zonă, aleasă aleator. De aceea, procesul de localizare a unei date în RAM este același pentru toate locațiile și cere un timp mediu egal, indiferent de poziția fizică în memorie a locației respective. De aici provine numele de *acces aleator*. Astăzi, și memoria ROM este cu acces aleator la informație, astfel că acronimul RAM caracterizează mai corect memoriile de tip *Read/Write*.

O structură de memorie RAM de tip $m \times n$ este un tablou de celule binare organizate în m cuvinte a câte n biți fiecare. Pentru $m=2^k$ avem un model standard de bloc de memorie RAM prezentat în figura următoare:



Comunicarea cu memoria RAM se face prin linii de date de intrare-ieșire, linii de adresă și linii de comandă (control) care precizează direcția de transfer a datelor (operația *Read* sau *Write*).

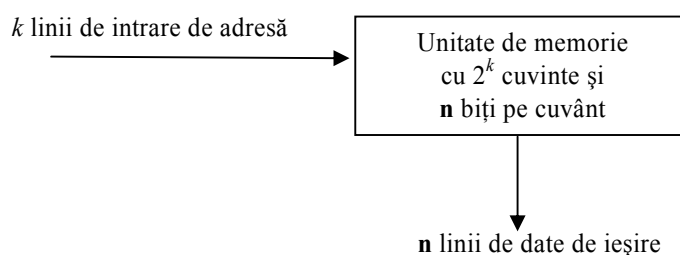
Cele k linii de adresă furnizează un număr binar pe k biți care precizează care cuvânt să fie ales din cei 2^k existenți în memorie. Cele două intrări de control (*Read* și *Write*) dau direcția de transfer a datelor solicitate.

În sistemul de calcul, o memorie de tip ROM este uzual folosită pentru memorarea programelor fixe (care nu se modifică în timpul funcționării SC) și a tabelor de constante care nu-și schimbă valoarea în sistem.

Într-o unitate de control, o memorie ROM poate fi folosită pentru conservarea informațiilor codificate care reprezintă secvențe de variabile interne de control necesare pentru activarea a diferite operații în sistem. O unitate de control care folosește ROM pentru memorarea informațiilor binare de control este numită *unitate de control cu microprogram* (*microprogrammed control unit*).

Memoria ROM nu are nevoie de linii de control de tip *Read* deoarece în fiecare moment cele n linii de ieșire furnizează automat cei n biți ai cuvântului selectat prin valoarea adresei de intrare. Mai mult, deoarece valorile biților în ROM au valori fixe, acest tip de memorie nu are nevoie de circuite proprii de stocare a datelor, așa cum întâlnim la RAM.

O structură de memorie ROM de tip $m \times n$ este un tablou de celule binare organizate în m cuvinte a câte n biți fiecare. Pentru $m=2^k$ avem o memorie ROM de tipul:



Tehnologia circuitelor integrate s-a impus cu succes și în construcția memoriilor, astfel încât spunem despre un ansamblu de bistabili dispuși într-o ordine bine precizată că formează un **cip de memorie**. Cele mai răspândite modele de cipuri sunt de tip SIMM (engl. *Single Inline Memory Module*) sau DIMM (engl. *Dual Inline Memory Module*). O placă de memorie SIMM sau DIMM este un grup de cipuri de memorie cablate pe o placă de dimensiuni reduse și vândute ca o singură unitate (memorie). Deosebirea dintre SIMM și DIMM constă în felul în care este plasată seria de conectori (pini de contact) prin care memoria respectivă se conectează pe placa de bază și anume dispunere pe o singură parte sau pe ambele părți ale plăcii de memorie. Fiecare conector are o funcție bine delimitată în transferul de date memorate, în adresare sau în controlul memoriei respective. O configurație tipică de SIMM este cu 8 cipuri de 32Mb (megabiți) fiecare, ceea ce înseamnă 32MB (megabytes) de memorie.

1.3.3 Structura ierarhică de organizare a memoriei

În timpul execuției lor, programele accesează în orice moment o zonă relativ mică a spațiului de adrese. Această observație a condus la delimitarea a două aspecte fundamentale în utilizarea memoriei, care sunt recunoscute ca **principiul localizării**. Astfel, avem **principiul localizării temporale**: *dacă este referită o entitate atunci este foarte probabil ca aceasta să fie referită din nou, în curând*; și **principiul localizării spațiale**: *dacă este referită o entitate atunci este foarte probabil ca „vecinii” ei să fie referiți în curând*.

Aplicarea principiului localizării a condus la organizarea ierarhică a memoriei. O **ierarhie de memorie** constă în nivele de memorii cu viteze de acces diferite și de dimensiuni diferite.

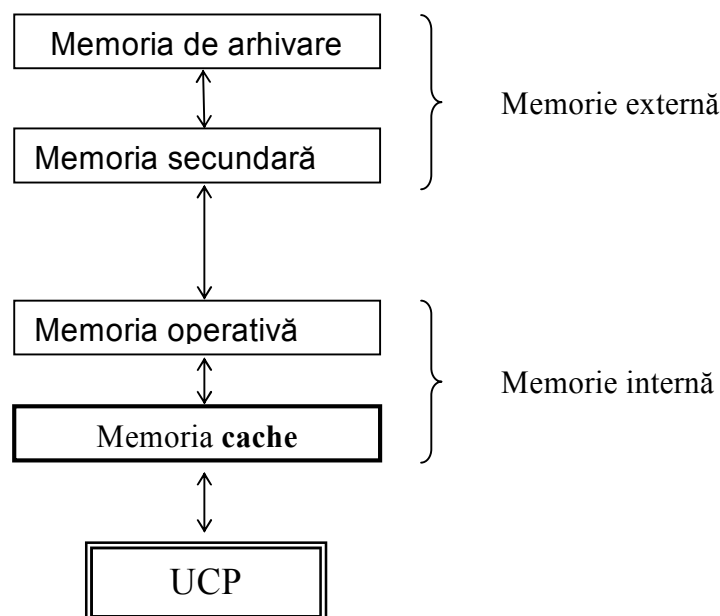
Din punctul de vedere al unui utilizator, memoria se împarte în *memorie externă* și *memorie internă*. Memoria externă este formată din toate componentele destinate stocării informației care sunt auxiliare sistemului de calcul. Cele mai folosite memorii externe sunt mediile de memorare bazate pe procedee optice (CD, DVD) și magnetice (hard-disc, dischetă). Toate celelalte componente de memorie ale sistemului de calcul formează memoria internă. Datele din memoria externă ajung să fie prelucrate de UCP numai după ce au fost transferate în memoria internă.

Din punctul de vedere al unui programator, informația stocată în memorie trebuie corelată cu programul curent de executat. Identificarea nivelelor de memorie trebuie să țină cont atât de tipul informației stocate (date propriu-zise sau programe), cât și de legăturile dintre informația stocată și programul curent în execuție la nivelul UCP. În consecință, pentru un programator, memoria este organizată ca *memorie secundară* și *memorie principală*. Memoria secundară (externă sau auxiliară) este de capacitate mare, practic nelimitată, conține informația păstrată pentru o utilizare ulterioară și este formată concret din periferice specializate. Memoria principală (internă sau operativă) este de capacitate fixă și conține informația în curs de prelucrare, adică programele în curs de execuție și datele programului curent. Memoria principală este conectată direct la magistrala sistem pentru a permite o viteză mare de transfer al informației.

Ținând cont de considerentele anterioare, memoria se poate prezenta ierarhic, pe nivele succesive de descriere, așa cum rezultă din figura următoare.

Săgețile de pe schema de mai sus arată că toate componentele de memorie comunică între ele în ambele sensuri. În plus, memoria cache comunică direct cu UCP.

Când o dată este solicitată de UCP, ea este căutată succesiv în zonele de memorie corespunzătoare nivelelor de mai sus: memorie cache, memorie operativă, memorie secundară, memorie de arhivare. Pentru regăsirea rapidă a datelor căutate, aceste memorii sunt construite astfel încât viteza de acces cea mai mare este la memoria cache și scade pentru celelalte tipuri, proporțional cu *distanța* față de UCP. În același timp, crește capacitatea de memorare, astfel că memoriile de arhivare ajung la capacități suficiente pentru stocarea de baze de date mari și foarte mari.



În continuare ne vom referi pe rând la tipurile de memorie din ierarhia de mai sus.

Memoria de arhivare sau memoria auxiliară este dată de dispozitivele de memorare care se conectează la sistemul de calcul (resurse informaționale). În memoria auxiliară se rețin programele între rulări. Acestea sunt gestionate de utilizator pentru folosirea eficientă a datelor depuse pe ele, de obicei fișiere și baze de date personale. Aceste suporturi de memorare sunt nevolatile, de obicei discuri magnetice (din 1965).

Ca entitate fizică a SC, memoria se mai numește **memorie reală**. Ca entitate de utilizare, vom numi memoria **virtuală**. Diferențele dintre acestea depind de modul de organizare fizică și de complexitatea memoriei în discuție.

Memoria virtuală este întotdeauna considerată ca o succesiune de octeți, în timp ce fizic, poziția a doi octeți vecini în memoria virtuală poate să difere. Legătura dintre un octet considerat în memoria virtuală și correspondentul său fizic este făcută de mecanismul de adresare.

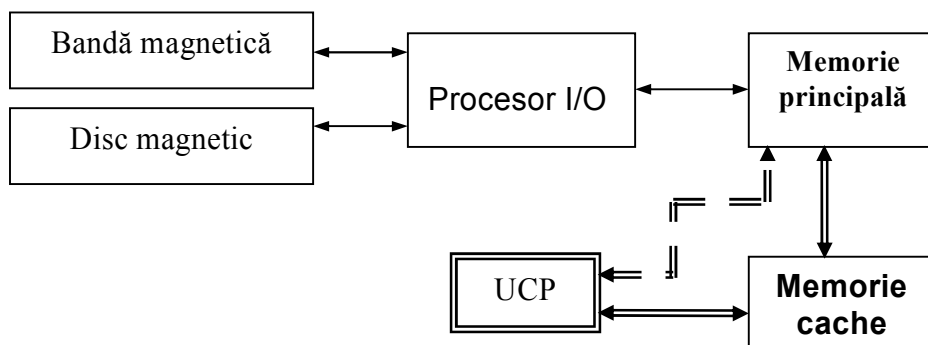
Memoria secundară apare la sistemele care cunosc mecanismul de memorie virtuală. Aceasta reprezintă o **extensie** a memoriei operative. Începând cu 1985, după ce a apărut sistemul 286 și modul de lucru protejat, calculatoarele IBM-PC au fost dotate cu **memorie extinsă** care permite o adresare naturală a spațiului de peste un megaoctet de memorie.

Memoria operativă conține datele și programele pentru toate procesele existente în sistem. O dată cu încheierea unui proces, automat se eliberează zona din memoria operativă alocată procesului respectiv.

Memoria cache (în *engleză, franceză* ascunzătoare, depozit) este memorie internă ultra-rapidă, care comunică direct cu UCP. Este de capacitate mică, dar cu viteză de acces foarte mare. În această zonă sunt reținute în fiecare moment datele cele mai recent utilizate de UCP. Data solicitată de UCP în prelucrarea curentă este adusă din zona de memorie în care a fost găsită și depusă în memoria cache *împreună* cu un număr de locații vecine datei solicitate, astfel încât, împreună, să umple memoria cache. Această metodă este recunoscută ca **principiul vecinătății** și este justificat astfel: față de data curent solicitată, este foarte probabil ca următoarea dată solicitată de UCP să fie depusă în memorie într-o locație apropiată de data curentă. O dată ce data curentă a fost adusă în cache împreună cu datele din locații alăturate, rezultă că următoarea regăsire se va face rapid, direct din zona cache. Acest principiu este strâns legat de **principiul localizării**.

În general, vom numi **memorie principală** acea zonă care comunică direct cu UCP. Ea va conține datele și programele frecvent utilizate de procesor. Memoria principală (primară) este volatilă, implementată cu memorii de tip DRAM.

Schema următoare precizează legăturile care se stabilesc între diferite componente ale sistemului de calcul implicate în mecanismul de transfer al datelor între UCP și memorii.



Din reprezentarea anterioară rezultă că dacă sistemul este recunoscut cu memorie cache atunci aceasta preia funcțiile memoriei principale, fiind direct conectată la UCP și realizând totodată organizarea transferului de informații între memoria principală și UCP.

Principalul rol al memoriei cache este de a compensa diferența între viteza de acces la memoria principală și viteza de prelucrare a procesorului. De obicei, procesorul este mai rapid decât memoria și, de aici, activitatea procesorului poate fi încetinită nedorit.

De asemenea, se remarcă prezența procesorului de intrare-ieșire cu rolul de a gestiona transferul datelor între memoria auxiliară și memoria principală. Se observă că memoria auxiliară nu are acces direct la UCP.

Astăzi există trei tehnologii de bază folosite în construcția ierarhiilor de memorii:

1. memoria principală este implementată cu DRAM-uri – memorii de viteză medie și capacitate medie;
2. nivelele mai apropiate de UCP (memoriile cache) folosesc SRAM-uri (*Static RAM*) – memorii de viteză mare și capacitate mică;
3. nivelele de memorii lente, dar de capacitate mare folosesc discuri magnetice.

Memoriile de tip SRAM sunt ceva mai scumpe per bit decât memoriile de tip DRAM, sunt mult mai rapide și consumă mai puțină energie electrică.

Tehnologia de memorare	Timp de acces standard	\$ / MB (1997)
SRAM	5 ns – 25 ns	100\$ – 250\$
DRAM	60 ns – 120 ns	5\$ – 10\$
Disc magnetic	10 ms – 20 ms	0,1\$ – 0,2\$

Uzual, nivelele de memorii interne realizate cu structuri de tip RAM sunt numite memorii RAM. Astfel, spunem că *memoria RAM* este memoria internă în care se efectuează prelucrările, în RAM se memorează cea mai mare parte a programelor, precum și datele care se modifică în timpul funcționării sistemului de calcul. Volatilitatea structurilor de tip RAM impune executarea de către programator a unei operații explicite de salvare a programului utilizator în curs de dezvoltare.

Între discurile magnetice și memoria principală există trei diferențe esențiale:

- ✓ discurile sunt nevolatile – deoarece se bazează pe tehnologii de memorare prin magnetizare (polarizare);
- ✓ discurile au acces lent – deoarece se bazează pe dispozitive mecanice;
- ✓ discurile sunt mai ieftine pe Mega-Byte – deoarece au capacitate de memorare mult mai mare, la un preț rezonabil; în plus, prețul de cost al unei secțiuni de disc este mai mic decât al unui circuit integrat (care stă la baza construcției componentelor memoriei principale, DRAM-uri).

La nivelul anului 1997, 1MB de disc era de 50 de ori mai ieftin decât 1MB de DRAM. Aceasta confirmă faptul că memoriile rapide sunt mai scumpe per bit decât memoriile lente, deci memoriile rapide sunt de obicei de capacitate mai mică.

1.3.4 Memorii cache

La prima mașină comercială care avea un nivel intermediar de memorie între UCP și memoria principală, acest nivel s-a numit **cache**. Astăzi, deși acesta este înțelesul propriu al cuvântului cache, termenul este folosit pentru a ne referi și la ORICE tip de memorie gestionat astfel încât să optimizeze accesul la locația căutată. Deci, **cache** denumește o metodă de obținere a unui acces rapid la porțiunile de cod și de date cele mai recent utilizate.

Memoria cache este organizată în blocuri de lungime fixă, numite linii de cache. O astfel de linie este o copie a unei zone din memoria din care a fost adusă informația în cache.

Proiectarea unei memorii cache trebuie să răspundă la întrebări de tipul:

- ✓ care este dimensiunea optimă a unei linii de cache?
- ✓ cum se regăsește informația conținută în cache?
- ✓ care linie se înlocuiește în cazul în care la un transfer memoria cache este plină?

În funcție de modurile de soluționare a acestor probleme, sunt recunoscute mai multe arhitecturi de cache, cum ar fi: memorii cache cu mapare directă, memorii cache asociative, memorii cache set asociative, memorii cache organizate pe sectoare.

În general, tehnicile prin care se implementează memoria cache și memoria virtuală sunt transparente pentru programele utilizator. Programele de aplicație sunt scrise ca și când ar lucra numai cu memoria principală.

Statistic, 90% din timpul rezervat pentru un program este utilizat pentru execuția numai a 10% din codul sursă al programului respectiv...

1.4. Proiectarea calculatoarelor moderne

După Patterson și Hennessy [5], se pot anunța patru principii de bază ale proiectării tehnologiei hardware, pe care le enumerăm în continuare.

Principiul 1. Simplitatea favorizează uniformitatea [5, pag.97].

Principiul 2. Mai mic înseamnă mai rapid [5, pag.99].

Principiul 3. Proiectarea bună necesită compromisuri bune [5, pag.108].

Principiul 4. Cazul frecvent trebuie făcut să aibă execuție rapidă [5, pag.133].

Rezumat.

Totalitatea componentelor destinate stocării datelor formează unitatea de memorie a sistemului de calcul. Acest paragraf detaliază aspecte concrete legate de structura și organizarea memoriei calculatoarelor moderne. Astfel, sunt abordate diferite puncte de vedere: (1) structura fizică a memoriei, (2) structura ierarhică de organizare, (3) structura internă dependentă de tipologia memoriei.

Cuvinte cheie.

bit, bit de informație, bistabil
locăție de memorie, octet, cuvânt
principiul localizării, principiul vecinătății, cache
memorie cu acces aleator, memorie read/write

Verificare.

1. Calculați capacitatea totală de memorare (în megabytes) a unei dischete cu două fețe și care este formatată cu următoarele constante: 80 piste (cilindri), 18 sectoare, 524 octeți pe bloc (un bloc constă într-un sector de pe o pistă). Capacitatea unui bloc este dată de 512 octeți de date propriu-zise și 12 octeți auxiliari (necesari marcării începutului blocului și localizării acestuia).
2. Uzual, o pagină de text are 40 linii cu 75 de caractere pe linie. Câte pagini de astfel de text se pot memora pe un CD de 600MB?
3. Folosiți principiile de organizare a memoriei cache pentru a justifica caracteristicile acesteia.
4. Care este capacitatea unei memorii cu 12 linii de intrare de adresă și 16 linii de ieșire (intrare) pentru date?
5. Incercați să justificați cele patru principii de bază ale proiectării hardware enunțate de Patterson și Hennessy.

NOTIȚE



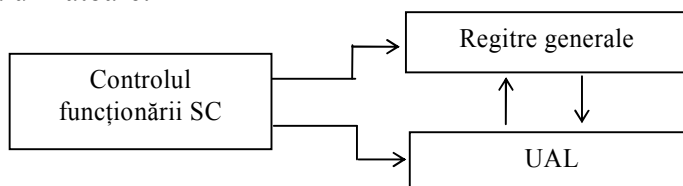
UNITATEA CENTRALĂ DE PRELUCRARE

CUPRINS

3.1.	Structura de bază a UCP -----	36
3.1.1.	Registrele generale -----	36
3.1.2.	Unitatea de comandă și control -----	37
3.1.3.	Calea datelor (engl., <i>datapath</i>) -----	38
3.2.	Organizarea stivei -----	42
3.3.	Funcționarea UCP. Execuția programelor -----	45
3.3.1.	Microoperație. Instrucțiune. Program -----	45
3.3.2.	Codurile instrucțiunilor -----	45
3.3.3.	Moduri de adresare -----	49
3.3.4.	Clasificarea instrucțiunilor -----	52
3.3.4.1.	Instrucțiuni pentru transferul datelor -----	52
3.3.4.2.	Instrucțiuni pentru gestiunea datelor -----	52
3.3.4.3.	Instrucțiuni pentru controlul programului -----	52
3.4.	Prelucrarea instrucțiunilor unui program -----	57
3.5.	Microprocesorul -----	58
3.5.1.	Caracteristici -----	58
3.5.2.	Setul de instrucțiuni ale microprocesorului -----	59
3.5.3.	Modelarea funcționării unui procesor -----	60
3.5.3.1.	Mașina cu trei adrese -----	60
3.5.3.2.	Mașina cu o adresă -----	62
3.5.4.	Structura internă de bază a unui microprocesor I 80x86 -----	65
3.6.	Multiprocesare -----	68

Componenta calculatorului care efectuează cea mai mare parte a operațiilor de prelucrare a datelor este numită **unitatea centrală de prelucrare**, prescurtat UCP (în engleză, CPU, *Central Processing Unit*).

Conform primelor două ecuații care definesc structura logică a sistemului de calcul ($UAL + UCC = UctrlP$ și $UctrlP + \text{Registre} = UCP$) rezultă că UCP constă din trei părți fundamentale, reprezentate în figura următoare:



În acest sistem unitar, rolul componentelor poate fi identificat astfel: (a) *registrele generale* sunt registrele de la nivelul UCP și memorează datele intermediare folosite în timpul executării instrucțiunilor; (b) *unitatea aritmetico–logică*, UAL, execută microoperațiile solicitate; (c) *unitatea de comandă și control*, UCC supervizează transferul informațiilor între registre și anunță UAL asupra următoarei operații de efectuat.

În acest capitol ne propunem: (1) să descriem modul în care se realizează comunicarea între componentele UCP prin intermediul liniilor de magistrală, (2) să explicăm mecanismul de lucru cu stiva, (3) să descriem principalele tipuri de formate de instrucțiune, modurile de acces la memorie și instrucțiunile care trebuie să existe în setul minimal al oricărui SC. În finalul capitolului vom fi în măsură să abordăm componenta fizică fundamentală a unui calculator, procesorul.

3.1. Structura de bază a UCP

O unitate centrală de prelucrare minimală constă din opt registre generale, o unitate de comandă și control, UCC și o unitate aritmetico–logică, UAL. Între aceste componente trebuie să existe linii de comunicare, care formează magistrale sau calea datelor (engl., *datapath*).

În acest paragraf vom lua în discuție aceste componente și mediul de comunicare dintre ele.

3.1.1. Registrele generale

- În sistemul de calcul, instrucțiunile de executat sunt memorate în locații succesive și sunt executate secvențial. Există o unitate de control care citește o instrucțiune de la o adresă precizată din memorie, o execută, apoi citește următoarea instrucțiune, o execută, ș.a.m.d.

Registrele generale au rolul unor locații rapide de memorie care, fie păstrează: date, rezultate intermediare, operanzi, alte informații necesare programului în execuție, fie efectuează operații elementare (microoperații).

Numărul și funcțiile registrelor generale ale unei UCP diferă în funcție de unitatea de prelucrare propriu-zisă. Structura minimală a registrelor generale ale unui sistem de calcul este dată de:

1	DR	<i>Data register</i>	Reține un operand memorat
2	AR	<i>Address register</i>	Reține o adresă de memorie
3	AC	<i>Accumulator</i>	Registru de prelucrare
4	IR	<i>Instruction register</i>	Reține cod de instrucțiune
5	PC	<i>Program counter</i>	Reține adresa unei instrucțiuni
6	TR	<i>Temporary register</i>	Reține date intermediare
7	INPR	<i>Input register</i>	Reține un caracter de intrare
8	OUTR	<i>Output register</i>	Reține un caracter de ieșire

Dacă registrul **DR** are 32 biți rezultă că în acest registru se vor putea reprezenta 2^{32} combinații binare, ceea ce înseamnă că sistemul respectiv lucrează cu date ale căror valori sunt mai mici decât $2^{32}-1$ (date reprezentabile pe 32 biți). Lungimea lui DR este egală cu lungimea cuvântului sistemului de calcul.

Dacă registrul **AR** are, de asemenea, 32 biți pentru a reprezenta o adresă de memorie atunci rezultă că în registrul AR se pot memora 2^{32} combinații binare care reprezintă adrese de memorie distincte. Mai mult, aceasta înseamnă că registrul AR poate adresa un spațiu de memorie de 2^{32} cuvinte de memorie.

Registrul acumulator **AC** este un registru general specializat în operațiile executate la nivelul UAL. Registrul AC păstrează unul din operanzi și rezultatul operației curente.

Registrul **IR** reține codul instrucțiunii în execuție.

Registrul **PC** reține adresa următoarei instrucțiuni care trebuie citită din memorie după ce instrucțiunea curentă va fi executată.

La fel ca și locațiile de memorie, registrele generale sunt identificate în codul de instrucțiune printrun *cod de adresă registru*. Spre deosebire de locațiile din memoria principală (unde, la un moment dat, se permite accesul la o singură locație), registrele generale pot fi accesate smultan.

3.1.2. Unitatea de comandă și control

Unitatea de comandă și control UCC execută secvențial instrucțiunile care compun programul curent al procesorului. Pentru a asigura **funcția de comandă**, UCC are următoarele componente:

1. **IR**, registru de instrucțiune (*Instruction Register*)
 - Păstrează codul instrucțiunii în curs de execuție
2. **PC**, registru numărător de program (*Program Counter*)
 - păstrează adresa următoarei instrucțiuni de executat a
3. **FR**, registru de stare (*Flag Register*)
 - Registrul de stare – păstrează în fiecare bit (flag) o informație predefinită despre contextul curent de lucru al procesorului
4. **BCC**, *Blocul Circuitelor de Comandă*
 - generează semnalele de comandă necesare execuției instrucțiunilor conform tipului de instrucțiune, fazei curente, informațiilor din FR;
 - toate microoperațiile și operațiile elementare care concură la execuția instrucțiunii curente sunt comandate prin semnale (microcomenzi) generate de BCC;
 - toate microoperațiile care se execută în același timp definesc o stare în execuția unei instrucțiuni, numită *fază*
5. **GT**, *Generatorul de Tact*
 - generează semnalul de tact pentru funcționarea sincronă a tuturor componentelor calculatorului;
 - frecvența generatorului de tact determină viteza de lucru a calculatorului;
 - cadența schimbării de stare pentru toate circuitele secvențiale sincrone din structura calculatorului este dată de GT
6. **GF**, *Generatorul de Faze*
 - creează succesiunea specifică de faze care compun o instrucțiune; faza următoare depinde de faza curentă, de tipul instrucțiunii și de informațiile de stare

Din enumerarea de mai sus rezultă că cronometrarea și sincronizarea pentru toate componentele sistemului de calcul este asigurată de o unitate (ceas) care generează impulsuri de tact. Dar, un impuls de ceas nu schimbă starea unui circuit sincron decât dacă aceasta este o

operație permisă conform cu un semnal de control. La rândul lui, semnalul de control este generat de **unitatea de control** a sistemului de calcul.

Există două tipuri de organizare a controlului: *control bazat pe circuite* și *control microprogramat*. Controlul bazat pe circuite presupune implementarea semnalelor de control prin porți logice, bistabili, decodare, ș.a. Fiecare comandă de control se regăsește în sistem prin starea circuitelor pe care le traversează. În organizarea microprogramată, informația de control este stocată în memorie. Această zonă de memorie este programată să inițializeze secvența de microoperații de executat. Fiecare comandă de control înseamnă o actualizare a programului memorat.

Funcția de control care specifică o microoperație este o variabilă binară, numită **variabilă de control**. Una din valorile binare va corespunde executării microoperației asociate, iar cealaltă valoare nu modifică starea regiștrilor din sistem. Într-un sistem bazat pe comunicare pe magistrale, semnalele de control care dau microoperațiile sunt grupuri de biți care selectează intrările în MUX, decodare și în componentele UAL.

Unitatea de control inițializează executarea secvențială pentru seturi de microoperații. La un moment dat, anumite microoperații au fost deja inițializate, în timp ce altele au rămas inactive.

În orice moment, valorile curente ale variabilelor de control se reprezintă printr-un șir de biți, numit **cuvânt de control**. Astfel, cuvintele de control pot fi programate să execute diferite operații asupra componentelor sistemului.

O unitate de control ale cărei variabile binare sunt stocate în memorie este o **unitate de control microprogramată**. Zona de memorie care păstrează variabilele de control se referă prin **memorie de control**. Fiecare cuvânt din memoria de control reprezintă o **microinstrucțiune**. O microinstrucțiune reunește una sau mai multe microoperații recunoscute în sistem. O secvență de microinstrucțiuni formează un **microprogram**.

Dacă, odată ce unitatea de control este activată, microprogramul nu se mai modifică atunci memoria de control poate fi o memorie de tip ROM. În acest caz, conținutul cuvântului de la o anumită adresă din această zonă de memorie reprezintă o microinstrucțiune.

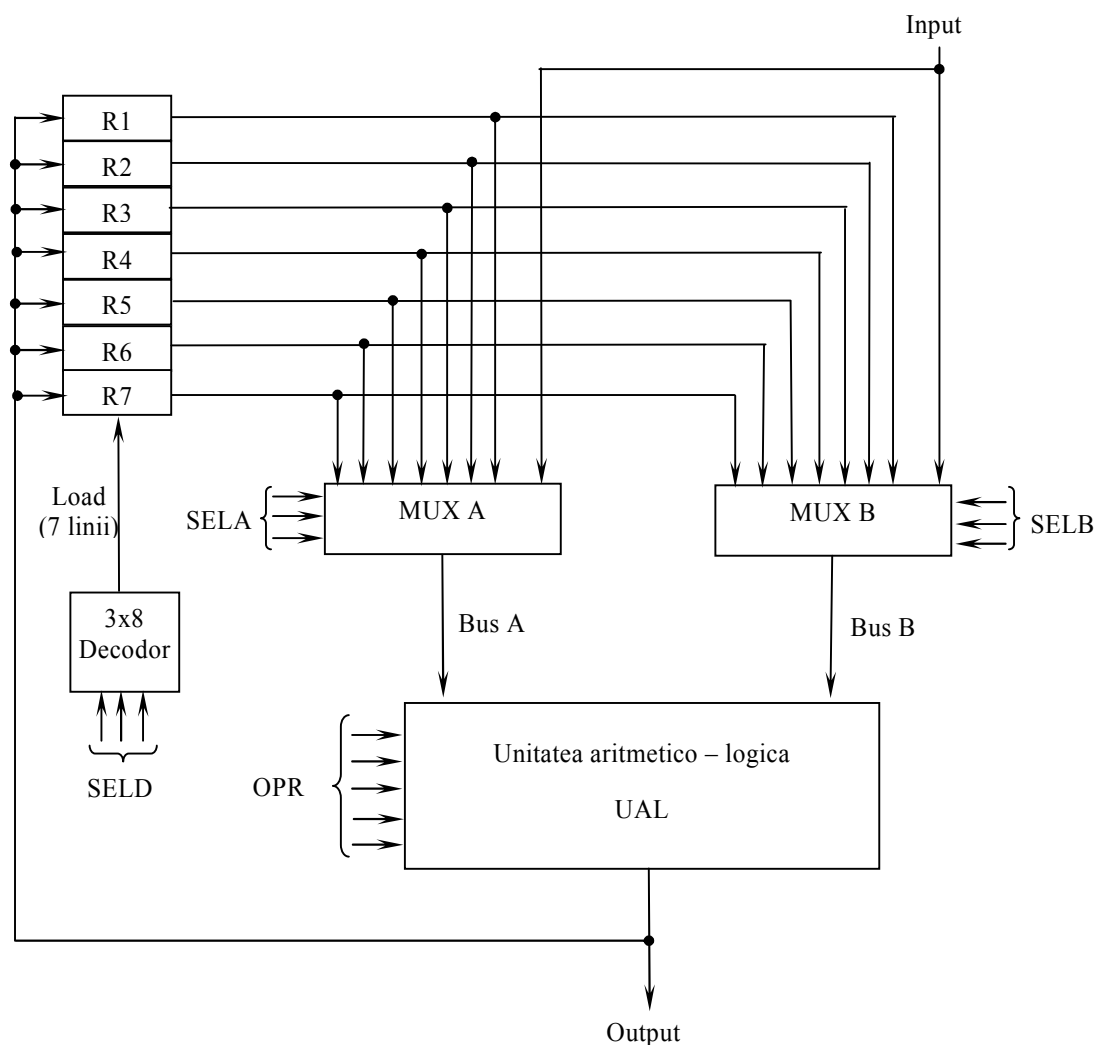
3.1.3. Calea datelor (engl., *datapath*)

Locațiile de memorie sunt necesare pentru a stoca pointeri, rezultate ale diferitelor numărări, adrese de revenire, rezultate intermediare, produse parțiale, ș.a.m.d. Pentru a scurta timpul de acces la zonele de memorie care păstrează aceste date, practic, acestea se memorează în **regiștrii de prelucrare** (*processor registers*). Când în UCP există un număr mare de regiștri, se recomandă ca aceștia să fie conectați prin linii de magistrală, de tip *common bus system*.

Organizarea unei magistrale pentru șapte regiștri UCP (conectați la UAL comună) ar putea fi cea de pe pagina următoare.

Ieșirea din fiecare registru este conectată la două circuite MUX pentru a forma cele două magistrale A și B. Liniile de adresă ale fiecărui MUX selectează un registru sau linia de date pentru una dintre magistrale. Datele de pe magistralele A și B reprezintă intrările în UAL comună. Operația selectată în UAL determină tipul microoperației care se va efectua. Rezultatul microoperației este disponibil pentru linia de date de ieșire, care este legată la intrările tuturor regiștrilor. Un decodor selectează registrul care va prelua informația de pe *Output* și activează intrarea de control *Load* pentru preluarea datelor în acel registru. Apoi, asigură transferul de date între *Output* și intrarea în registru destinație selectat.

Notăm: SELA – intrarea de selecție pentru MUX A, SELB – intrarea de selecție pentru MUX B, OPR – selectorul de operație pentru UAL și SELD – selectorul registrului destinație pentru decodor.



În acest exemplu avem 14 intrări de selecție în circuit: 3 pentru SELA, 3 pentru SELB, 3 pentru SELD și 5 pentru OPR. Combinația valorilor pe aceste intrări formează **cuvântul de control** (*control word*). Acesta se reprezintă astfel:

3	3	3	5
SELA	SELB	SEL	OPR

Cei trei biți de pe intrările SELA selectează registrul sursă pentru intrarea A a UAL, iar SELB selectează registrul sursă pentru intrarea B a UAL. Cei trei biți ai SELD selectează registrul destinație folosind decodorul și cele 7 ieșiri de validare a încărcării, *Load*. În concluzie, cuvântul de control aplicat pe intrările de selecție precizează o microoperație, conform cu următoarele tabele:

Conținutul pentru	Registrul selectat de		
	SELA	SELB	SELD
000	Input	Input	nimic
001	R1	R1	R1
010	R2	R2	R2
011	R3	R3	R3

100	R4	R4	R4
101	R5	R5	R5
110	R6	R6	R6
111	R7	R7	R7

Conținutul pentru OPR	Operația	Simbolul
00000	Transfer A	TSFA
00001	Increment A	INCA
00010	Add A + B	ADD
00101	Subtract A –	SUB
00110	Decrement A	DECA
01000	AND A și B	AND
01010	OR A și B	OR
01100	XOR A și B	XOR
01110	Complement	COMA
10000	Shift right A	SHRA
11000	Shift left A	SHLA

Rezumat.

În acest paragraf sunt enumerate principalele registre care intră în structura unui calculator elementar. Detalierea unor aspecte legate de structura și rolul în sistem a acestor registre va face obiectul considerentelor din paragrafele următoare.

O parte importantă a acestui paragraf o constituie exemplul de transfer de date din regiștri pe magistrală. Pentru acest exemplu studenții trebuie să poată justifica structura fiecărei componente folosite (MUX, registru).

Acest paragraf conține, de asemenea, un exemplu de *datapath* (cale a datelor) pentru comunicare între UAL și regiștri. Scopul acestui exemplu este dublu: pe de o parte să familiarizeze studenții cu un ansamblu complex de componente logice care include și linii de comunicare (magistrale) și, pe de altă parte, să exemplifice conceptul de **cuvânt de control** (structură și utilizare). Acest al doilea obiectiv este important în perspectiva înțelegerii *codului de instrucțiune*, respectiv a formatului de descriere a programului de executat la nivelul procesorului.

Verificare.

- Pentru exemplul de cale a datelor (engl. *datapath*) prezentat anterior, răspundeți la următoarele cerințe:
 - justificați tipul decodului 3x8;
 - cum se modifică structura de date dacă UAL reprezintă codul operațiilor pe 7 biți?
 - este posibil să conectăm la o astfel de magistrală 3 regiștri; cum se modifică dimensiunile componentelor logice în acest caz?
 - dimensiunea regiștrilor este determinată? Depind alte dimensiuni de dimensiunea regiștrilor?
- Pe modelul *datapath* anterior, se cere:
 - să se aleagă valori concrete pentru conținutul celor șapte regiștri pe patru biți;
 - să se execute următorul program de control:

01010001100010
10001111101100
00111000110000
11100111100001

NOTIȚE

3.2. Organizarea stivei

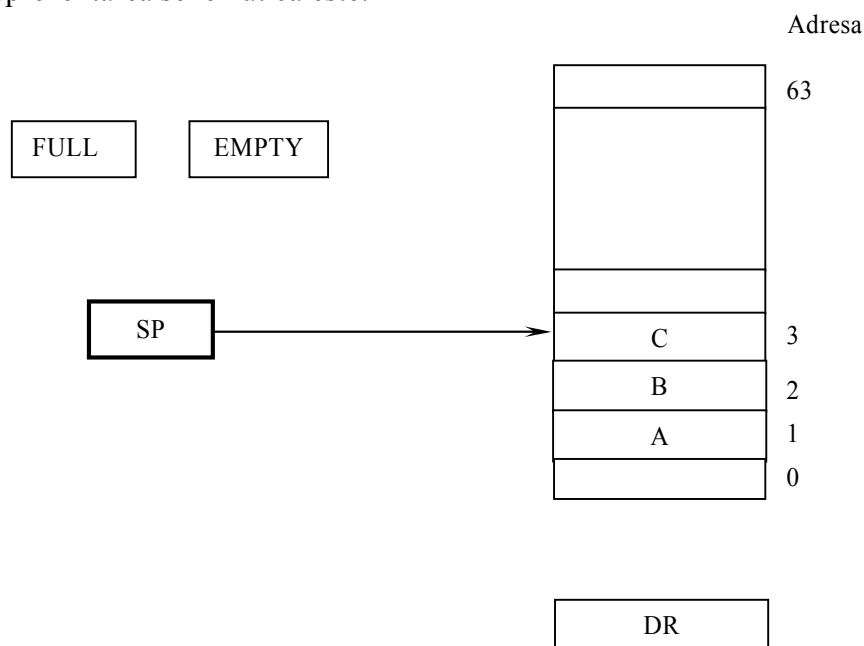
O structură importantă care este inclusă în unitatea centrală de prelucrare la majoritatea SC este cea de stivă sau listă de tip LIFO. Aici, prin **stivă** vom înțelege o unitate de memorare care se manipulează astfel încât ultima dată introdusă este prima care se prelucrează.

Intr-un sistem de calcul digital, stiva este în esență o unitate de memorie cu un registru de adresă SP care execută o singură operație, anume cea de contorizare (numărare), după ce a fost încărcat cu o valoare inițială. Registrul care păstrează adresa pentru stivă se numește **pointer de stivă** (*stack pointer*, **SP**). Valoarea acestuia întotdeauna *poartează* spre entitatea considerată la momentul curent în vârful stivei. Dimensiunea registrului SP depinde de capacitatea stivei, ceea ce înseamnă, cu alte cuvinte, că dimensiunea registrului SP trebuie să fie suficientă pentru a reprezenta toate adresele din spațiul de adresă corespunzător stivei.

Operațiile care se pot efectua asupra unei stive sunt cele de **inserare** și **ștergere**. Inserarea se mai numește *push*, iar ștergerea, *pop*.

Operațiile specifice pe stiva unui SC nu sunt inserarea și ștergerea datelor ci aceste operații sunt simulate prin incrementarea și decrementarea valorii din registrul SP.

Stiva SC poate fi depusă într-o zonă contiguă de memorie sau poate fi organizată ca o colecție finită de cuvinte de memorie sau de regiștri. Pentru un registru de stivă cu 64 cuvinte de memorie, reprezentarea schematică este:



Registrul SP conține o valoare binară egală cu adresa cuvântului care este la momentul respectiv în vârful stivei.

În acest exemplu avem trei entități în stivă: A, B și C. C este în vârful stivei, astfel că în SP avem valoarea 3.

Pentru a șterge vârful stivei, se efectuează o operație *pop* pe stivă prin citirea cuvântului de memorie de la adresa 3 și decrementarea conținutului lui SP. Astfel, SP memorează valoarea 2, adică *poartează* la B și conținutul B este considerat în vârful stivei. Practic, C a fost citit, nu mai este în vârful stivei, dar nu a fost eliminat din stivă.

Pentru a insera un element în stivă se efectuează o operație *push* prin incrementarea valorii lui SP și scrierea cuvântului la următoarea adresă liberă.

Pentru o stivă de 64 cuvinte de memorie, SP are 6 biți deoarece $64 = 2^6$. Astfel, SP nu poate memora o valoare mai mare decât $63 = 111111$. Când 63 este mărit cu o unitate rezultatul este 0 deoarece

$$111111 + 1 = 1000000$$

și SP reține biții cel mai puțin semnificativi, adică 000000.

Analog, când 000000 este decrementat cu 1, rezultatul este 111111.

Pentru analizarea acestor situații particulare, se folosesc doi regiștri speciali, fiecare de 1 bit, numiți FULL și EMTY. Registrul FULL este setat pe 1 când stiva este plină, iar EMTY este setat pe 1 când stiva este vidă.

Registrul de date DR reține ceea ce se va scrie în stivă la următoarea operație *push* sau ceea ce s-a citit din stivă la ultima operație *pop*.

Funcționarea stivei constă în precizarea operațiilor elementare pentru *push* și *pop*.

Inițial, SP este șters, deci are valoare 0, EMTY este setat pe 1 și FULL este setat pe 0. Astfel, SP pointează cuvântul de la adresa 0.

Dacă stiva nu este plină atunci este permisă o operație *push* prin:

```
(SP) <— (SP) + 1
[SP] <— (DR)
IF (SP) = 0 THEN FULL <— 1
EMTY <— 0
```

Dacă stiva nu este goală atunci este permisă o operație *pop* prin:

```
(DR) <— [SP]
(SP) <— (SP) – 1
IF (SP) = 0 THEN EMTY <— 1
FULL <— 0
```

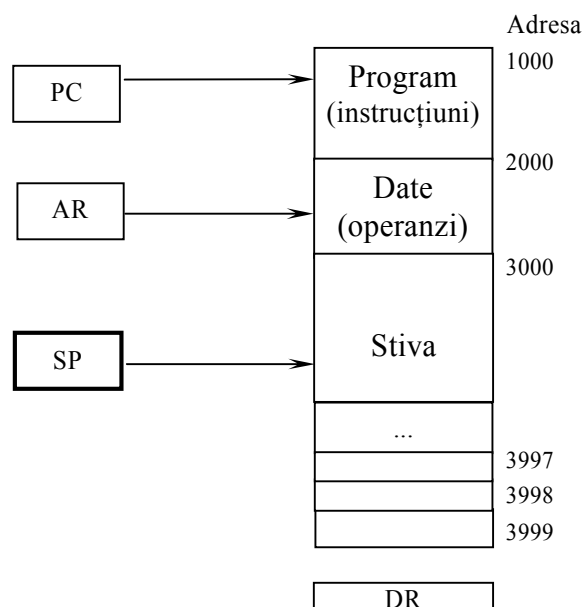
Observație. Prin SP ne referim la adresa registrului SP, prin (SP) ne referim la conținutul registrului SP, iar prin [SP] ne referim la conținutul de la adresa conținută de registrul SP.

Memoria rezervată pentru stivă poate fi o unitate individuală sau poate fi implementată într-un RAM atașat UCP. Implementarea unei stive în UCP se face prin asignarea unei zone de memorie pentru operațiile pe stivă și prin folosirea unui registru de prelucrare care să îndeplinească funcțiile lui SP.

În reprezentarea de pe pagina următoare memoria SC este împărțită în trei segmente: segment de program, segment de date și stiva.

Registrul PC, *Program Counter*, pointează la adresa următoarei instrucțiuni din programul de executat. Registrul de adresă AR pointează un șir de date. SP pointează spre vârful stivei. Cei trei regiștri sunt conectați la o magistrală comună de adrese și numai unul poate depune o adresă în memorie.

Registrul PC este folosit pentru a citi o instrucțiune de executat. AR este folosit în execuție pentru a citi un operand. SP este folosit pentru a simula inserarea sau ștergerea unui element din stivă.



Cele mai multe SC nu au componente hardware care să verifice situațiile de operare pe stivă în configurații improprii, cum ar fi depășirile de tip **overflow**, pentru stivă plină și **underflow**, pentru stivă vidă. Limitele stivei pot fi verificate prin mecanisme soft, de exemplu prin folosirea a doi regiștri de prelucrare: unul va memora adresa cea mai mică a cuvintelor de memorie rezervate stivei, iar celălalt adresa cea mai mare. Cu acestea, după o operație *push*, SP se va compara cu primul registru, iar după o operație *pop*, cu valoarea din al doilea registru.

Observație. În exemplul stivei pe 64 cuvinte de memorie, o inserare în stivă corespunde unei operații de mărire a adresei la care se încarcă conținutul lui DR. În exemplul următor, în care am reprezentat stiva pe o zonă de memorie de la adresa 4001 la 3000, o inserare în stivă corespunde unei micșorări a adresei de memorie la care se încarcă conținutul registrului DR. De acest aspect trebuie ținut cont în scrierea instrucțiunilor care asigură utilizarea stivei, începând chiar cu valoarea cu care se inițializează SP. Aceasta trebuie să fie adresa rezervată zonei de memorie de la baza stivei.

Avantajul lucrului cu stiva constă în faptul că UCP poate referi memoria stivei fără să fie nevoie să precizeze o anumită adresă, deoarece adresa respectivă este întotdeauna disponibilă și este automat actualizată în registrul de stivă, SP.

Rezumat.

Stiva, ca structură de date, face parte din tematica cursurilor de programare. În ceea ce privește arhitectura calculatoarelor, interesează stiva sistemului de calcul ca zonă de memorie rezervată și organizată pe principiul stivei (LIFO). De asemenea, ca și în cazul paragrafului precedent, conceptul este esențial pentru înțelegerea modului în care procesorul (implicit, programatorul prin programul pe care îl scrie în limbaj de asamblare) lucrează cu cele trei segmente de memorie: stivă, date, cod.

În particular, pentru lucru cu segmentul de stivă, trebuie reținut rolul registrului SP în efectuarea operațiilor pe stivă (încărcare, descărcare).

Cuvinte cheie.

stiva sistemului de calcul
registrul SP, operații specifice: încărcare, descărcare
eroare de depășire

Verificare.

Dacă stiva de memorie a unui sistem de calcul are 256 de cuvinte, care trebuie să fie capacitatea minimă a registrului SP în acest caz?

Ce se poate spune despre rezultatul unei operații *push* pe stivă dacă în urma efectuării ei conținutul registrului SP este nul?

NOTIȚE

3.3. Funcționarea UCP. Execuția programelor

3.3.1. Microoperație. Instrucțiune. Program

Organizarea hardware internă a unui SC este bine definită dacă se specifică:

mulțimea de regiștri și funcțiile lor;
secvența de microoperații de executat pentru informațiile binare depuse în fiecare registru;

instrucțiunea care inițializează fiecare secvență de microoperații.

Pentru specificarea secvențelor de operații de transfer de informații între regiștri se folosește un simbolism, numit **limbaj de transfer pe regiștri** (*register transfer language*).

Un SC digital este capabil să execute diferite microoperații și, în plus, poate primi instrucțiuni pentru precizarea următoarei secvențe de microoperații care trebuie executată.

Pentru SC, o **instrucțiune** este un cod binar care precizează o secvență de microoperații de executat de către calculator.

Un **program** este un set de instrucțiuni care precizează operațiile, operanzii și legăturile dintre aceste elemente în execuție.

Există o mare varietate de limbaje de programare pe care utilizatorul le poate folosi pentru a scrie un program pentru calculator, în timp ce sistemul de calcul poate executa un program numai dacă este reprezentat intern în binar. De aceea, un program scris în orice limbaj de programare trebuie întâi *translatat* (tradus) în reprezentare binară.

Programele scrise pentru un calculator se pot încadra într-una dintre următoarele categorii:

1. **cod binar**, dat de o secvență de instrucțiuni și operanzi binari care reproduce exact modul de reprezentare în memorie a secvenței respective. Un astfel de program se va numi **program în cod mașină** sau **program în limbaj mașină**;
2. **cod octal** sau **hexazecimal**, dat de o conversie a codului binar în octal sau hexa;
3. **cod simbolic**, dat de o reprezentare cu simboluri a codurilor instrucțiunilor. Fiecare instrucțiune dată prin simboluri poate fi translatată într-o instrucțiune în cod binar. Această traducere este făcută de un program special, numit **assembler**. Deoarece un *assembler* traduce simboluri, aceste programe simbolice se numesc *programe în limbaj de asamblare*;
4. **limbaje de programare de nivel înalt**. Un program care traduce un astfel de program în binar se numește **compiler**.

Rezumat.

Acest paragraf definește conceptele de **instrucțiune** și **program**, în perspectiva paragrafelor ulterioare și, în plus, pentru înțelegerea noțiunii de **cod** – ca mod de reprezentare a unui program. Astfel, în funcție de cunoștințele și de preferințele programatorului, acesta poate opta pentru una dintre variantele de scriere a codului sursă al programului său: cod binar, cod octal, cod simbolic sau cod în limbaj de programare de nivel înalt.

3.3.2. Codurile instrucțiunilor

Pentru un SC cu control programat, calculatorul citește fiecare instrucțiune din memorie și o depune într-un *registru de control*. Unitatea de control interpretează acest cod și determină execuția lui, prin inițializarea unei secvențe de microoperații.

Fiecare calculator posedă un set propriu de instrucțiuni pe care le poate executa. Capacitatea de a memora și de a executa instrucțiunile, precum și conceptul de *program memorat* sunt proprietățile principale ale unui calculator de uz general.

Codul unei instrucțiuni este dat de un grup de biți care dirijează calculatorul pentru a executa o anumită operație. De obicei, acesta este delimitat în două câmpuri, fiecare având o semnificație bine precizată: zona rezervată codului de operație și zona de adresă.

Cea mai mare parte a codului formează zona operațională, numită **codul operației** (*operation code*). Acesta definește tipul operației, de exemplu: adunare, scădere, înmulțire, deplasare, complementare. Numărul de biți rezervați pentru codul operației depinde de numărul total de instrucțiuni ale SC analizat.

Observație. Este bine să precizăm aici care este diferența între o operație (*computer operation*) și o microoperație. O operație este o parte dintr-o instrucțiune stocată în memorie. Ea este dată de un cod binar care o identifică în sistemul de calcul. Unitatea de control preia instrucțiunea din memorie și îi interpretează biții care reprezintă codul operației. Apoi, generează o secvență de semnale de control pentru a inițializa microoperațiile corespunzătoare în regiștri interni ai SC. Pentru fiecare cod de operație, unitatea de control generează o secvență de microoperații necesare executării operației specificate. De aici, codul operației se numește de multe ori **macrooperație** deoarece specifică un set de microoperații.

Revenind, codul operației reprezintă operația de executat. Această operație trebuie să lucreze cu date memorate în regiștrii de prelucrare sau direct în memorie. De aceea, codul unei instrucțiuni trebuie să precizeze, nu numai operația, ci și informații cu privire la operanzi. Aceste informații formează **zona de adresă**, care conține regiștrii sau cuvintele de memorie unde se găsesc operanzii și registrul sau cuvântul de memorie în care se depune rezultatul. Registrele de prelucrare se regăsesc prin încărcarea în zona de adresă a unui cod binar de k biți care să identifice în mod unic unul dintre cei 2^k regiștri din sistem. Cuvintele de memorie se dau în codul instrucțiunii prin adresele lor.

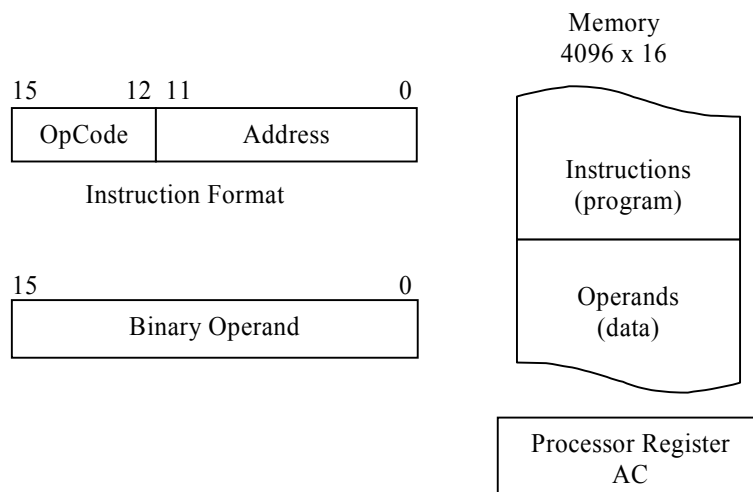
Există numeroase modalități de a organiza codul binar al unei instrucțiuni. Fiecare SC are propriul format al codului de instrucțiune. Acesta este conceput de designer-ul calculatorului, adică de cel care precizează arhitectura SC. Lungimea codului de instrucțiune diferă de la un calculator la altul și nu este neapărat aceeași pentru toate instrucțiunile.

În continuare dăm trei exemple de organizare a formatului instrucțiunilor, construite mai mult pe considerente didactice decât cu referire la anumite sisteme de calcul.

Exemplul 1.

Cel mai simplu mod de a organiza un calculator este de a folosi un registru de prelucrare și un cod de instrucțiune format din *OpCode* și o adresă. Această adresă de memorie indică unității de control unde găsește operandul pe care trebuie să-l folosească. Acest operand va fi citit din memorie și valoarea rezultată va fi operată cu conținutul registrului de prelucrare.

Acest tip de organizare este prezentat în figura următoare:



Instrucțiunile sunt depuse într-o zonă de memorie distinctă față de cea în care sunt depuse datele. Aici avem o memorie cu $4096 = 2^{12}$ cuvinte, deci adresele lor se vor reprezenta pe 12 biți.

Dacă codul unei instrucțiuni se reprezintă pe 16 biți atunci rezervăm 12 biți pentru adresa unui operand și rămân 4 biți pentru codul operației, *OpCode*. Astfel, acest sistem va recunoaște un set de $16 = 2^4$ operații.

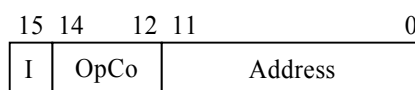
Unitatea de control citește cei 16 biți ai codului unei instrucțiuni din zona de memorie rezervată programului. Cei 12 biți ai zonei de adresă a codului identifică în zona de date locația din care să se citească operandul pe 16 biți necesar. Apoi, unitatea de control inițializează operația specificată de cei 4 biți ai *OpCode*. Aceasta se execută pe actualul conținut al registrului de prelucrare (notat aici cu AC) și pe valoarea operandului citit din memorie.

Observație. Uneori este util să folosim zona de adresă a unei instrucțiuni nu cu semnificația unei adrese ci ca valoare a operandului de citit. Când partea a doua a unui cod de instrucțiune reprezintă un operand, spunem că instrucțiunea are **operand transmis prin valoare**. Când zona de adresă a unui cod de instrucțiune conține adresa la care se găsește în memorie operandul necesar, spunem că **operandul este transmis prin adresă** (directă).

O a treia posibilitate este ca la adresa din zona de adresă a codului instrucțiunii să se găsească adresa de memorie a operandului. În acest caz, **operandul este transmis prin adresă indirectă**.

Exemplul 2.

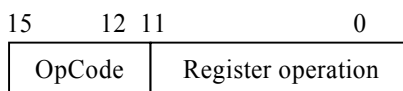
Pentru a face diferența între adresele directe și indirecte se poate rezerva în formatul instrucțiunii un bit modal (*de mod*) care va avea o anumită valoare binară dacă adresa este indirectă și valoarea complementară dacă adresa este directă. Acesta se notează de obicei cu I (*indirect address mode bit*) și se reprezintă ca în figura următoare:



Exemplul 3.

Dacă o operație nu are nevoie de operand din memorie atunci zona de adresă din codul instrucțiunii poate fi folosită în alte scopuri, de exemplu pentru a preciza operații suplimentare recunoscute de sistem. Se obține astfel o instrucțiune cu referință la un registru de prelucrare (*register – reference instruction*).

În acest caz, în zona de adresă a codului instrucțiunii nu mai avem o adresă ci o operație sau un test de efectuat asupra registrului de prelucrare.



Rezumat.

Conținutul acestui paragraf este central în contextul capitolului al treilea. Aici se definește **codul de instrucțiune** cu cele două câmpuri ale sale: zona de cod al operației și zona de adresă. Zona de cod al operației definește operația de executat, în timp ce zona de adresă se referă la operanzii implicați în efectuarea operației respective: date, adrese, modul de calcul al adresei. Zona de adresă poate să conțină: (1) nici o adresă; (2) o adresă; (3) două adrese (registru-registru, registru-memorie sau memorie-memorie); (4) trei adrese (adresele operanzilor și adresa rezultatului, caz în care fiecare adresă poate referi un registru sau o locație de memorie).

Exemplele care se dau în acest paragraf sunt esențiale pentru înțelegerea modurilor de adresare. Astfel, fiecare exemplu reprezintă o situație particulară, concretă, pentru combinațiile

de valori (semnificații) pe care le pot avea cele două zone ale codului de instrucțiune: zona de cod și zona de adresă.

De asemenea, exemplele din acest paragraf anticipează formatul de instrucțiune folosit de mașina cu trei adrese, respectiv de mașina cu o adresă (vezi *Modelarea funcționării unui procesor*).

Verificare.

1. Pentru exemplul 1 anterior, care este lungimea minimă a cuvântului pentru o memorie de 1024 cuvinte?
2. Refaceți calculele de la exemplul 1 în cazul unei memorii de 64 cuvinte de 8 biți fiecare. Câte operații se pot defini pe o astfel de structură?

NOTIȚE

3.3.3. Moduri de adresare

În codul unei instrucțiuni, OpCode specifică operația de executat. Această operație trebuie executată asupra unei date salvate în memorie sau într-un registru. Felul în care este regăsit operandul depinde de conținutul zonei de adresă a codului instrucțiunii, deci de **modul de adresare** pentru fiecare instrucțiune. Acesta specifică o regulă pentru interpretarea și modificarea câmpului adresă a codului instrucțiunii înainte ca operandul să fie referit.

Pentru a înțelege modurile de adresare trebuie cunoscută secvența de operații pe care le efectuează unitatea de control pentru fiecare instrucțiune în parte. Aceasta constă din următoarele etape:

1. preluarea instrucțiunii din memorie (faza de **extragere**, engl., *fetch*),
2. decodificarea instrucțiunii (faza de **decodificare**, engl., *decode*),
3. executarea instrucțiunii (faza de **execuție**, engl., *execute*).

În calculator există un registru special, PC, care păstrează *urma* instrucțiunilor programului memorat. PC reține adresa instrucțiunii care se va executa și este incrementat la fiecare preluare a unei instrucțiuni din memorie.

Etapă de decodificare identifică operația de executat, modul de adresare a instrucțiunii și localizează operandul.

Apoi, calculatorul execută instrucțiunea respectivă și revine la pasul 1 pentru a prelua următoarea instrucțiune de executat.

Modul de adresare se poate regăsi în codul instrucțiunii pe un bit sau pe mai mulți. Într-un sistem de calcul se poate ca modul de adresare să difere de la o instrucțiune la alta sau, chiar pentru o aceeași instrucțiune adresarea să fie combinată.

A. Pentru regăsirea operandului, există două moduri de adresare care nu folosesc o adresă în câmpul al doilea al codului instrucțiunii. Acestea sunt **modul implicit** și **modul direct**.

În modul implicit de adresare se presupune că operandii sunt dați prin definiția instrucțiunii, fără ambiguitate. În modul direct de adresare, operandul apare în codul instrucțiunii, în câmpul de adresă. Un exemplu de astfel de instrucțiuni sunt cele de inițializare a conținutului unui registru cu o anumită valoare.

B. Când câmpul de adresă al codului de instrucțiune specifică un registru de prelucrare, spunem că instrucțiunea este în mod registru. În acest caz operandul se află în registrul specificat și acesta va fi selectat din lista de identificare a regiștrilor sistemului.

Un caz particular de adresare cu registru este situația în care în registrul referit se află adresa cuvântului de memorie care reține operandul căutat. Spunem atunci că **adresarea este indirectă**.

Particularizând în continuare, putem avea o instrucțiune care incrementează (decrementează) valoarea din registru la fiecare accesare a memoriei. În acest caz avem **adresare indirectă în mod registru cu autoincrementare** (autodecrementare). Un exemplu pentru această situație sunt regiștrii care referă o tabelă de date depusă în memorie.

Câmpul de adresă al codului de instrucțiune este folosit de unitatea de control pentru a obține valoarea operandului memorat. Uneori, valoarea dată în acest câmp este adresa operandului (vezi situațiile anterioare), dar poate fi și adresa la care se determină (calculează) adresa operandului. Spunem că avem **adresa absolută a operandului** dacă ea reprezintă adresa de memorie la care se găsește operandul căutat.

În exemplele anterioare am anticipat modul de adresare directă și modul de adresare indirectă. În concluzie, putem spune că la adresarea directă adresa efectivă a operandului apare în câmpul de adresă al instrucțiunii. La adresarea indirectă, câmpul de adresă al instrucțiunii cuprinde adresa la care este stocată în memorie adresa efectivă a operandului.

C. În alte situații, modul de adresare impune ca valoarea memorată în câmpul de adresă să fie adunată cu conținutul unui anumit registru pentru a obține adresa efectivă a

operandului căutat. Un astfel de registru implicat în calcul de adrese poate fi PC, un **registru de index** sau un **registru de bază**.

Dacă este vorba de registrul PC atunci avem **adresare relativă**. Concret, când conținutul din câmpul de adresă este adunat cu conținutul registrului PC se obține o adresă efectivă a cărei poziție în memorie este relativă la adresa următoarei instrucțiuni de executat (reținută în PC).

Exemplu. $M[PC]=825$, $address=24$. Instrucțiunea de la locația 825 este citită din memorie și $PC \leftarrow PC + 1 = 826$. Rezultă că se calculează adresa $826 + 24 = 850$, care reprezintă a-24-a locație de memorie de la adresa următoarei instrucțiuni de executat.

Avantajul acestei adresări este faptul că adresa relativă se reprezintă pe o secvență de biți mai scurtă decât adresa efectivă.

Dacă pentru a obține adresa efectivă se adună conținutul câmpului de adresă cu conținutul unui registru de index atunci avem o **adresare indexată**. Un registru de index este un registru de prelucrare special care conține o valoare de ordine sau de indexare (indiciere). Aici, codul instrucțiunii cuprinde adresa de început a unui șir de date memorat și fiecare dată din șir se regăsește în memorie relativ la adresa de început. Diferența între adresa de început și adresa efectivă a operandului este indexul (indicele) memorat în registrul de index.

Dacă pentru a obține adresa efectivă se adună conținutul câmpului de adresă cu conținutul unui registru de bază atunci avem o **adresare cu registru de bază**. Diferența față de adresarea indexată constă în tipul registrului implicat în calculul de adresă. Aici, registrul de bază conține o adresă de bază și codul instrucțiunii dă deplasamentul relativ la adresa de bază respectivă.

Execuția repetată a secvenței *extrage-decodifică-execută* aplicată pe instrucțiunile unui program oarecare arată că un program poate fi executat de către un alt program, care va fi pentru primul un **interpretor**. Din acest punct de vedere execuția unui program se poate face prin hardware, caz în care procesorul este cel care execută efectiv programul, sau prin software, în cazul execuției prin interpretor. Această echivalență între procesoarele hardware și interpretoare are consecințe foarte importante pentru structurarea și proiectarea sistemelor de calcul. Astfel, după specificarea limbajului mașină L al unui nou calculator, echipa de proiectare poate decide dacă va construi un procesor hardware care să execute direct programele scrise în limbajul L sau dacă va scrie un interpretor care să interpreteze programele scrise în L (adică să le traducă în limbajul mașină al unui procesor hardware deja existent). Sunt posibile și construcții hibride, cu o parte de execuție hardware și o parte de interpretare software.

Extragerea instrucțiunilor din memorie este un punct critic în viteza de execuție a acestora. Pentru a rezolva această problemă, calculatoarele aveau posibilitatea de a extrage în avans instrucțiuni din memorie, astfel încât acestea să fie rapid disponibile atunci când erau necesare. Aceste instrucțiuni erau păstrate într-un set de registre numit **tamponul pentru extragere în avans**. În principiu, extragerea în avans împarte execuția instrucțiunii în două etape: extragerea și execuția propriu-zisă. Conceptul **benzii de asamblare a procesorului** (engl. *pipeline*) extinde strategia: în loc să se împartă execuția instrucțiunii în doar două părți, de multe ori aceasta este împărțită în mai multe etape, de fiecare etapă ocupându-se o componentă hardware dedicată acesteia, toate etapele putând să fie executate în paralel. De exemplu, o bandă de asamblare cu 5 segmente (etape) ar putea fi delimitată astfel:

1. unitatea de extragere a instrucțiunii;
2. unitatea de decodificare a instrucțiunii;
3. unitatea de extragere a operanzilor;
4. unitatea de execuție a instrucțiunii;
5. unitatea de scriere a rezultatului.

Rezumat.

În funcție de conținutul concret al fiecăreia dintre zonele din codul de instrucțiune, în programul propriu-zis al procesorului, se deosebesc modurile de adresare. În acest paragraf am dat o clasificare a modurilor de adresare în trei categorii:

(A) adresare *fără adresă*

- ✓ adresare implicită
- ✓ adresare imediată (prin valoare)

(B) adresare cu adresă

- ✓ adresare directă (absolută) – cu memoria
- ✓ adresare registru
- ✓ adresare indirectă

(C) adresare cu registru de calcul

- ✓ adresare relativă
- ✓ adresare indexată – procesoarele Intel au două registre de index: SI și DI, ambele cu autoindexare postinstrucțiune
- ✓ adresare cu registru de bază – procesoarele Intel au două registre de bază pentru date: BX și BP; de asemenea, registrele de segment CS, DS, ES și SS au rol de registre de bază pentru segmentele pe care le reprezintă

Cuvinte cheie.

mod de adresare
banda de asamblare a procesorului

Verificare.

1. Care este diferența dintre un interpretor și un compilator?
2. Identificați cele cinci etape ale *pipeline*-ului intern pentru un program de control scris pentru exemplul de *datapath* de la începutul capitoului.

NOTIȚE

3.3.4. Clasificarea instrucțiunilor

Deși setul de instrucțiuni diferă de la un calculator la altul, toate sistemele de calcul trebuie să aibă un ansamblu minimal de instrucțiuni. Acestea se regăsesc în următoarele trei categorii:

- A. instrucțiuni pentru transferul datelor – asigură transferul datelor dintr-o locație în alta, fără modificarea conținutului;
- B. instrucțiuni pentru manipularea datelor – efectuează operațiile aritmetice, logice și cele de deplasare;
- C. instrucțiuni de control al programului – cu rol în luarea deciziilor și în dirijarea execuției microprogramului.

În particular, unele sisteme de calcul au în plus instrucțiuni pentru operații de transfer pe regiștri și / sau instrucțiuni care permit utilizatorului să intervină asupra microprogramului.

3.3.4.1. Instrucțiuni pentru transferul datelor

Cele mai uzuale transferuri se realizează:

- ✓ între memorie și regiștrii de prelucrare (instrucțiunea **Load** sau **Store**);
- ✓ între regiștrii de prelucrare și stivă (instrucțiunea **Push** sau **Pop**);
- ✓ între regiștrii de prelucrare și regiștrii de intrare / ieșire (instrucțiunea **Input** sau **Output**);
- ✓ între doi regiștri de prelucrare (instrucțiunea **Move** sau **Exchange**).

În funcție de conținutul zonei de adresă a codului, instrucțiunea poate fi apelată cu diverși parametri. Cunoașterea și utilizarea acestora țin de limbajul de asamblare folosit și de sistemul de calcul pe care se lucrează.

3.3.4.2. Instrucțiuni pentru gestionarea datelor

Această clasă de instrucțiuni reprezintă posibilitățile de calcul ale sistemului. Din această categorie fac parte:

1. instrucțiunile aritmetice;
2. instrucțiunile logice și cele de prelucrare la nivel de bit;
3. instrucțiunile de deplasare.

În general, calculatoarele *cunosc* operațiile aritmetice de adunare, scădere, înmulțire și împărțire. Este posibil ca unele sisteme să folosească numai adunarea și înmulțirea sau chiar numai adunarea și să simuleze și celelalte operații. În funcție de operanzii pe care poate lucra o **instrucțiune aritmetică**, aceasta poate avea diferite forme. De exemplu, nu sunt puține calculatoarele care admit trei sau chiar mai multe instrucțiuni pentru adunare. Acestea pot fi: ADDI, pentru operanzi binari – întregi, ADDF, pentru operanzi în virgulă flotantă sau ADDD, pentru operanzi zecimali.

Instrucțiunile logice efectuează operații binare pe șiruri de biți memorati în regiștri. Fiecare bit este considerat individual și tratat ca o variabilă booleană. Operațiile binare fundamentale sunt: setare pe 0, setare pe 1 și complementare.

La rândul lor, **instrucțiunile de deplasare** pot fi de tip logic, de tip aritmetic sau de rotație. De exemplu, deplasarea logică este utilă atunci când trebuie completate cu 0 pozițiile libere ale unei valori care nu ocupă întregul registru. Deplasarea aritmetică asigură fixarea pe poziție corectă a bitului de semn. Rotația reprezintă o deplasare circulară, adică un bit *eliminat* la o deplasare la dreapta este încărcat în bitul cel mai semnificativ eliberat.

3.3.4.3. Instrucțiuni pentru controlul programului

Instrucțiunile sunt întotdeauna memorate în locații de memorie succesive. Ori de câte ori o instrucțiune este preluată din memorie în vederea executării ei, conținutul registrului PC este

incrementat, astfel încât în fiecare moment el conține adresa următoarei instrucțiuni de executat. După executarea unei instrucțiuni de transfer de date sau de prelucrare de date, **controlul** revine la programul care dă succesiunea instrucțiunilor de executat.

Uneori, execuția programului de control poate schimba adresa conținută de PC și astfel, să modifice succesiunea inițială de instrucțiuni de control de executat. Cu alte cuvinte, instrucțiunile de control programat precizează și condițiile în care poate fi modificat conținutul registrului PC, dacă instrucțiunile de transfer sau de prelucrare solicită o astfel de modificare. Schimbarea valorii reținute în PC ca rezultat al executării unei instrucțiuni de control determină o pauză (*break*) în secvența de instrucțiuni de executat. Aceasta este o idee importantă pentru funcționarea sistemelor digitale deoarece determină controlarea fluxului de instrucțiuni, chiar cu posibilitatea de a executa bucle sau un segment de program auxiliar.

Cele mai importante instrucțiuni ale unui program de control sunt: **Branch, Jump, Skip, Call, Return, Compare și Test**.

Instrucțiunile **Branch** și **Jump** sunt similare și determină încărcarea în PC a adreselor transmise ca parametri. În acest fel se asigură că următoarea instrucțiune se va executa de la adresa precizată, deci se poate realiza un salt în secvența de instrucțiuni predefinită. Aceste două instrucțiuni de control pot fi condiționate sau nu.

Instrucțiunea **Skip** determină salt peste instrucțiunea curentă, fără ca aceasta să fie executată. De asemenea, și această instrucțiune poate fi condiționată sau nu.

Instrucțiunile **Call** și **Return** se folosesc în contextul lucrului cu subrutine și vor fi descrise ulterior.

Instrucțiunile **Compare** și **Test** modifică indirect secvența instrucțiunilor de executat. Astfel, ele execută operații specifice de scădere și respectiv SI logic. Rezultatul aplicării lor nu se reține, dar acesta modifică în mod corespunzător un anumit bit din registrul de stare.

Registrul de stare. Atunci când s-a convenit ca circuitul UAL să fie completat cu un registru de stare, biții acestuia vor fi utilizați în analiza contextului curent, înainte sau după executarea anumitor instrucțiuni. Biții registrului de stare se numesc **biți de stare**, sau **biți de condiționare a codului** (*condition-code bits*), sau **flag-uri**.

În figura următoare avem un registru de stare pe 4 biți, notați cu C (*Carry*), S (*Sign*), Z (*Zero*) și V (*Overflow*). Acești biți sunt setați (puși pe 1) sau reseați (puși pe 0) în urma executării unei operații în UAL.

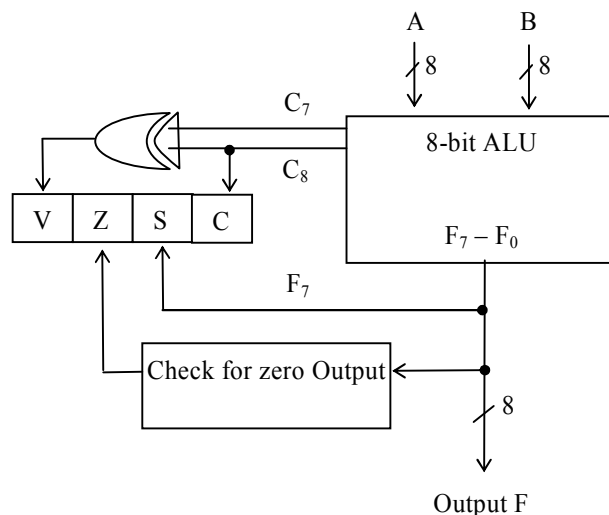
Semnificația celor patru biți de stare este următoarea:

C = 1 dacă bitul de transport C_8 este 1;

S = 1 dacă bitul de ordin maxim (bitul de semn) F_7 este 1;

Z = 1 dacă rezultatul furnizat de UAL este zero, adică conține numai zerouri;

V = 1 dacă s-a obținut o depășire. Aceasta se interpretează diferit, în funcție de reprezentarea operanzilor în complement față de 2 sau față de 1.



Apelul și revenirea din subrutine. O subrutină este o secvență de instrucțiuni care efectuează o acțiune de sine stătătoare, bine delimitată. În timpul executării unui program, o subrutină poate fi apelată de mai multe ori pentru a fi executată în diferite puncte ale programului principal. De fiecare dată când este apelată o subrutină, este executată automat o instrucțiune *branch* de deviere spre prima instrucțiune din subrutină pentru a iniția executarea instrucțiunilor subrutinei. După executarea tuturor instrucțiunilor subrutinei, se execută o instrucțiune *branch* de revenire în programul principal.

Instrucțiunile care transferă controlul programului înspre și dinspre subrutină pot fi: *call subroutine*, *jump to subroutine*, *branch to subroutine* sau *branch and save address*.

Apelul unei subrutine constă în două operații:

adresa următoarei instrucțiuni (cea reținută de PC) este memorată într-o locație temporară astfel încât subrutina să știe unde să revină; aceasta va fi adresa de revenire – *return address*;

controlul este transferat la începutul subrutinei.

Ultima instrucțiune a oricărei subrutine, *return from subroutine*, realizează **revenirea în programul principal** prin transferarea adresei de revenire din locația temporară în PC.

Locația de memorie temporară pentru adresa de revenire diferă de la un SC la altul. Unele SC folosesc pentru aceasta prima locație de memorie a zonei rezervată subrutinei, altele folosesc o locație fixă din memorie, altele un registru de prelucrare, iar altele folosesc o stivă de memorie.

Varianta cea mai eficientă este cea cu stivă de memorie. În acest caz, fiecare apel de subrutină înseamnă o operație *push* pe stivă cu adresa de revenire din subrutina respectivă și fiecare revenire înseamnă o operație *pop* de pe stivă pentru adresa respectivă. Avantajul apare la apelul în cascadă pentru mai multe subrutine deoarece organizarea memoriei ca stivă asigură că revenirea se face întotdeauna corect, adică în programul apelant (în ultimul program care a apelat procedura respectivă).

Astfel, o instrucțiune de apel constă în:

$SP \leftarrow SP - 1$

$M[SP] \leftarrow PC$

$PC \leftarrow \text{adresa efectivă a primei instrucțiuni a subrutinei}$

Dacă o altă subrutină este apelată din subrutina în execuție atunci o nouă adresă de revenire este depusă pe stivă, s.a.m.d.

O instrucțiune de revenire va consta în:

$PC \leftarrow M[SP]$

$SP \leftarrow SP + 1$

Folosind stiva pentru lucrul cu subrutine, programatorul nu mai trebuie să se concentreze asupra corectitudinii revenirii din subrutină în programul apelant.

Un al doilea avantaj al memorării adreselor de revenire pe stivă apare la lucrul cu subrutine recursive. Pe stivă nu interesează care adresă este depusă la un moment dat. Dacă sistemul ar lucra cu altă variantă de memorare a adreselor de revenire atunci ar trebui avut în vedere că este foarte posibil ca recursivitatea să se piardă prin distrugerea adresei de revenire memorate anterior pentru aceeași subrutină.

Intreruperea unui program. O **întrerupere** este o excepție sau un semnal intern sau extern transmis SC, prin care acestuia i se anunță apariția unui eveniment neobișnuit, adică neconform cu secvența de instrucțiuni în curs de execuție. Atunci când evenimentul s-a produs, au loc, în ordine, următoarele trei acțiuni:

1. suspendarea programului în curs de desfășurare;
2. lansarea în execuție a unui program de depanare (*service program*), adică a unei rutine specializate, numită **Rutină de Tratare a Intreruperii (RTI)** sau **Handler de întrerupere**, care deservește întreruperea;
3. reluarea execuției programului suspendat.

Cauzele producerii acestor evenimente pot fi externe sau interne.

Exemple de evenimente de natură externă:

- * apăsarea unei anumite combinații de taste, de exemplu CTRL/BREAK;
- * terminarea unei operații de intrare/ieșire desfășurată în *background* (adică în timp ce utilizatorul urmărește desfășurarea altei acțiuni);

De cele mai multe ori, în urma tratării întreruperii externe, se reia activitatea programului suspendat.

Exemple de întreruperi de natură internă:

- * împărțirea la zero;
- * tentativa de adresare a unei zone de memorie care nu există;
- * tentativa de execuție a unei instrucțiuni având un cod inexistent;
- * depășirea capacității de reprezentare a rezultatului unui calcul.

De cele mai multe ori, în urma tratării întreruperii interne, controlul este transmis SO, fără a mai relua activitatea programului suspendat.

Întreruperile externe sau interne sunt însoțite de semnale ale componentelor hardware ale SC. Față de acestea, în SC pot apărea și evenimente inițiate de executarea unor instrucțiuni. Acestea vor fi *întreruperi software*. Corespunzător, avem rutine care deservește apariția întreruperilor interne sau externe, care vor fi activate la apariția evenimentului, fără a se putea prevedea momentul apariției. Vom spune despre *handler*-ele din această categorie că *sunt activate prin evenimente*. În schimb, cealaltă parte a *handler*-elor oferă aceleași servicii ca o bibliotecă de subprograme, apelabile de către programele utilizator. Activarea lor are loc atunci când sunt solicitate prin program. *Handler*-ele din această categorie *sunt activate prin instrucțiuni speciale de apel de întreruperi*.

La apariția unei întreruperi SC trebuie, în ordine:

- 1) să determine tipul evenimentului care a generat întreruperea (intern, extern);
- 2) să afle care este sursa (cauza) întreruperii;
- 3) să determine adresa RTI (a rutinei de tratare a întreruperii), adică în ce manieră se poate identifica *handler-ul* corespunzător.

Rezolvarea acestor probleme presupune că pentru fiecare tip de eveniment și pentru fiecare cauză posibilă a unei întreruperi să existe câte un *handler* specific. *Metoda cea mai folosită* pentru localizarea rapidă a RTI este **vectorizarea întreruperilor**. Aceasta constă în a asocia, pentru fiecare întrerupere, o locație de memorie cu adresă fixă. În această locație fixă se trece adresa RTI corespunzătoare întreruperii. Deci, în momentul apariției unei întreruperi, mai întâi se preia din locația de memorie asociată întreruperii adresa *handler*ului corespunzător ei și apoi se cedează controlul de execuție *handler*-ului.

O parte dintre rutinele de tratare a întreruperilor sunt furnizate o dată cu sistemul de calcul. O altă parte sunt scrise de către proiectanții sistemului de operare. În sfârșit, utilizatorii pot să-și scrie propriile lor RTI.

În proiectarea unui *handler* de întrerupere trebuie respectate, în ordine, următoarele etape:

- 1) Salvarea contextului programului suspendat: regiștri, flag-uri, stiva etc. De obicei, SC reține contextul curent prin memorarea conținutului registrului PC și a cuvântului PSW – *program status word*. Acesta caracterizează UCP deoarece conține valorile curente ale biților de stare.

- 2) Interzicerea tratării întreruperilor care pot perturba evoluția normală a handler-ului respectiv. De exemplu, handler-ul care rezolvă împărțirea la zero poate să conțină, eventual din greșeală, o împărțire la zero. În acest caz ar trebui să nu fie apelat din nou același handler.
- 3) Rezolvarea situației care a generat întreruperea, aceasta constituind partea esențială a handler-ului.
- 4) Ridicarea restricțiilor impuse la etapa 2.
- 5) Refacerea contextului salvat la etapa 1.
- 6) Predarea controlului către programul care a fost suspendat sau către sistemul de operare.

Regula unanim acceptată de către proiectanții sistemelor de întreruperi este aceea că orice întrerupere este luată în considerare numai între execuțiile a două instrucțiuni mașină succesive. Niciodată UCP nu răspunde unei întreruperi până când nu a încheiat execuția instrucțiunii curente. Înainte de fiecare fază de preluare a următoarei instrucțiuni din memorie, unitatea de control verifică și rezolvă toate semnalele de întrerupere. Dacă apar simultan două întreruperi, circuitele hard ale SC trebuie să decidă care dintre ele va fi deservită prima.

Rezumat.

În acest paragraf sunt prezentate principalele instrucțiuni recunoscute de procesor într-un program de control. Acestea sunt grupate în funcție de rolul lor în sistem:

- instrucțiuni pentru transferul datelor
 - ✓ transfer între memorie și regiștri de prelucrare
 - ✓ transfer între regiștri de prelucrare și stivă
 - ✓ transfer între doi regiștri de prelucrare
- instrucțiuni pentru gestionarea datelor
 - ✓ instrucțiuni aritmetice
 - ✓ instrucțiuni logice și de prelucrare la nivel de bit
 - ✓ instrucțiuni de deplasare
- instrucțiuni de control al programului.

Cuvinte cheie.

instrucțiune
registru de stare
subrutină
întrerupere, vectorizarea întreruperilor, tratarea întreruperilor

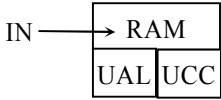
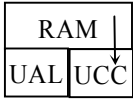
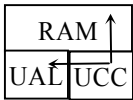
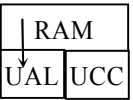
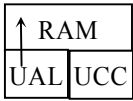
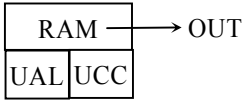
Verificare.

1. Ce este o subrutină? De ce este potrivită utilizarea stivei pentru gestionarea apelului și revenirii dintr-o subrutină?
2. Ce este o întrerupere? Ce se înțelege prin *vectorizarea întreruperilor*?

NOTIȚE

3.4. Prelucrarea instrucțiunilor unui program

Prelucrarea instrucțiunilor unui program de către UCP constă în principal în următoarele etape:

1.	UCC emite semnale de comandă către dispozitivul periferic de intrare pentru transferul de instrucțiuni și de date către memoria RAM;	
2.	UCC extrage instrucțiunile din memoria internă și le interpretează (decodifică);	
3.	în funcție de interpretarea făcută, UCC emite semnale de comandă către UAL și RAM;	
4.	datele auxiliare solicitate din RAM sunt transferate în UAL, unde se execută, sub supravegherea UCC, operația aritmetică sau logică decodificată;	
5.	rezultatele obținute în UAL sunt dirijate de UCC către RAM;	
6.	se repetă etapele de la 1. la 5. pentru fiecare instrucțiune;	
7.	După executarea tuturor instrucțiunilor, UCC direcționează RAM să transfere rezultatele către un dispozitiv periferic de ieșire.	

Verificare.

Identificați în structura fizică a sistemului de calcul unitățile funcționale corespunzătoare componentelor implicate în acest paragraf.

NOTIȚE

3.5. Microprocesorul

Indiferent de structura sa internă, UCP este cea mai importantă componentă pentru funcționarea unui SC. De aceea este numită și *creierul sistemului de calcul*. Deoarece UCP se concretizează fizic în componente numite **procesoare**, rezultă de aici că procesorul (în diversele sale variante) este unitatea de prelucrare a sistemului, deci componenta fundamentală a unui sistem de calcul.

La supercalculatoare și la calculatoarele medii-mari, UCP poate fi construită prin circuite structurate pe un cadru principal, numit **mainframe**, în timp ce, la microcalculatoare, UCP este construită din circuite integrate structurate pe pastile de siliciu care vor forma **microprocesoarele**. Un microprocesor este o UCP realizată într-un singur circuit integrat.

3.5.1. Caracteristici

Puterea unui microcalculator este dată de:

- ✓ viteza de execuție,
- ✓ memorie și
- ✓ acuratețea circuitelor electronice componente.

Un microprocesor se apreciază în funcție de:

- ✓ viteza cu care execută instrucțiunile și
- ✓ cantitatea de memorie internă pe care o poate accesa în mod direct.

Viteza de execuție a unui microprocesor depinde, la rândul ei, de:

- ✓ lungimea cuvântului și
- ✓ viteza ceasului.

Lungimea cuvântului reprezintă capacitatea registrului de bază, adică numărul de biți ce poate fi prelucrat la un moment dat de microprocesor. Această valoare este invers proporțională cu viteza de execuție. Primele microprocesoare pe 8 biți au fost create de firma ZILOG și numite Z80. Cronologic, urmează microprocesoarele din familia Intel care au lungimea cuvântului de 16 biți (I 8086, I 8088, I 80286), de 32 biți (I 80386, I 80486) sau de 64 biți pentru microprocesoarele Pentium.

Viteza ceasului sau **frecvența de tact** reprezintă numărul de impulsuri electronice pe care le produce ceasul intern al microprocesorului într-o secundă. Viteza ceasului este măsurată în Hz și multiplii „informatici” ai acestora. Actualmente, microprocesoarele produse au depășit viteza ceasului de 1,5 GHz.

Memoria internă pe care o poate accesa în mod direct depinde de lungimea cuvântului. Astfel, pentru cuvânt de 16 biți microprocesorul va putea accesa 2^{16} octeți, în timp ce pentru un cuvânt de 32 biți, microprocesorul va putea accesa 2^{32} octeți.

Deoarece este componenta fundamentală a SC, microprocesorul este cel care determină și denumirea calculatorului gazdă. Prezentăm în continuare microprocesoarele din familia Intel I 80x86 și sistemele care le includ.

La apariția sa în 1978, procesorul 8086 era cu adevărat o unitate de prelucrare cu un singur cip pe 16 biți. Proiectul lui 8086 era oarecum similar cu cel al lui 8080, dar nu integral compatibil. Cipul 8086 a fost urmat de cipul 8088, care avea aceeași arhitectură ca și 8086 și rula aceleași programe, dar avea o magistrală pe 8 biți și nu pe 16 biți, fiind deci mai lant, dar mai ieftin decât 8086. acest cip a devenit rapid standard pentru industria calculatoarelor personale odată cu alegerea sa de către IBM pentru producerea sistemului inițial IBM PC.

Nici 8088, nici 8086 nu puteau adresa mai mult de 1MO de memorie. Astfel, la începutul anilor '80, Intel a proiectat 80286, o versiune îmbunătățită, compatibilă cu 8086. următorul pas a fost un adevărat procesor pe 32 biți și anume 80386, compatibil cu procesoarele anterioare. Această compatibilitate era oun plus pentru cei care doreau să ruleze în continuare vechile soft-uri și un inconvenient pentru cei care ar fi preferat o arhitectură simplă, modernă, curată, fără impedimentele greșelilor și tehnologiile trecutului.

Încercând să răspundă cerințelor pieței, în 1989 Intel scoate procesorul 80486: o versiune mai rapidă a lui 80386, cu coprocesorul matematic inclus și cu o memorie cache de 8KB inclusă pe cip. Procesorul 486 mai avea incorporat și suportul pentru multiprocesare, pentru a permite producătorilor de calculatoare să construiască sisteme cu mai multe UCP.

Pentru generația următoare de procesoare, Intel a avut surpriza să descopere că, de fapt, numerele (ca 80486) nu pot fi mărci înregistrate. Astfel, următoarea generație de procesoare se va numi Pentium, de la grecescul πππππ cinci. Spre deosebire de 80486 care avea o singură bandă de asamblare internă (*pipeline*), Pentium are două astfel de benzi, deci este de ori mai rapid. Următoarea generație de procesoare este Pentium Pro, o rupere clară față de trecut pentru că noul Pentium Pro avea o organizare internă total diferită și putea să execute până la cinci instrucțiuni simultan. În plus, memoria cache internă avea două nivele: 8KB pentru instrucțiuni și 8KB pentru date pe un nivel și încă 256KB pe al doilea nivel.

În continuarea seriei, noul procesor Intel a fost Pentium II, în fapt un Pentium Pro cu extensii speciale pentru multimedia (MMX, *MultiMedia eXtension*). Aceste instrucțiuni aveau rolul de a accelera calculele necesare pentru procesarea semnalelor audio și video, înlăturând astfel nevoia unor coprocesoare speciale multimedia.

La începutul anului 1998, Intel a introdus o linie nouă de produse numită Celeron – o variantă a procesorului Pentium II ieftină, cu performanțe scăzute, destinată PC-urilor de duzină.

3.5.2. Setul de instrucțiuni ale microprocesorului

Un aspect important al arhitecturii unui SC este dat de setul de instrucțiuni recunoscute de microprocesorul acelui sistem. De aceste instrucțiuni depinde construcția și folosirea limbajului mașină al calculatorului în discuție.

La început, datorită posibilităților reduse ale echipamentelor hardware, calculatoarele lucrau cu instrucțiuni puține și simple. Ulterior, apariția circuitelor integrate a permis mărirea numărului de instrucțiuni recunoscute de un microprocesor, ajungându-se la sisteme cu 100 sau chiar 200 de instrucțiuni în setul de bază. Implicit, aceste SC pot prelucra un număr mare de tipuri de date și diverse moduri de adresare a memoriei.

În 1980, un grup de la Berkeley, condus de David Patterson și Carlo Séquin, a început să proiecteze cipuri VLSI pentru UCP fără interpretor. Ei au introdus termenul RISC pentru acest concept și și-au numit cipul de unitate centrală RISC I, urmat la scurt timp de RISC II. Puțin mai târziu, în 1981, la Stanford, John Hennessy a proiectat și fabricat un cip întrucâtva diferit, pe care l-a numit MIPS. Aceste cipuri au evoluat în produse comerciale importante, procesoarele SPARC și respectiv MIPS. De fapt, acronimul MIPS este pentru *Milioane de Instrucțiuni pe Secundă*, procesorul MIPS fiind denumit astfel numai printr-un joc de cuvinte. Ulterior, structura internă a acestui procesor a permis și o denumire potrivită: *Microprocessor without Interlocked Pipeline Stages*.

Majoritatea microprocesoarelor folosite în prezent fac parte din familia CISC (Complex Instruction Set Computer). Cele mai performante microprocesoare de acest tip sunt **Pentium**-urile firmei Intel și **M68040** al firmei Motorola, ultimul echipând PC-urile de tip Macintosh.

Tendința actuală se îndreaptă spre microprocesoarele de tip RISC (Reduced Instruction Set Computer). În general, acestea lucrează pe 64 biți, cu o viteză de până la 200 MHz. La baza arhitecturii RISC stă intenția unor constructori de microprocesoare de a reduce timpul de execuție a unui program prin simplificarea setului de instrucțiuni. Aceștia au recomandat ca SC să folosească mai puține instrucțiuni, care să aibă structuri simple, astfel încât să poată fi executate mai rapid în UCP, adesea fără un necesar de memorie suplimentară.

Procesoarele MIPS apar ca un caz particular de procesoare RISC și sunt folosite astăzi în special pe stațiile de lucru Silicon Graphics.

Pe scurt, caracterizarea celor două arhitecturi ar fi:

CISC	<ol style="list-style-type: none"> 1. număr mare de instrucțiuni (100 – 250); 2. unele instrucțiuni execută acțiuni specializate (se apelează rar); 3. număr mare de moduri de adresare a memoriei (5 – 20); 4. formatul instrucțiunilor nu are lungime fixă; 5. există instrucțiuni care prelucreză operanzi direct în memorie;
RISC	<ol style="list-style-type: none"> 1. relativ puține instrucțiuni; 2. relativ puține moduri de adresare a memoriei; 3. există acces limitat la memorie pentru instrucțiunile <i>Load</i> și <i>Store</i>; 4. toate operațiile se efectuează în regiștrii UCP; 5. instrucțiunile au format fix (sunt ușor de decodificat); 6. control bazat pe circuite (preferabil controlului microprogramat).

3.5.3. Modelarea funcționării unui procesor

În acest paragraf vom prezenta principiile generale de funcționare ale unui procesor, considerând două modele, alese pe considerente didactice să reprezinte simplu și pe înțeles procesorul. Această modalitate de prezentare a rezultat în urma unui proces de simplificare a construcției unui procesor. De aceea, în practică nu se pot regăsi aceste modele foarte simple, dar ele constituie scheletul oricărui model de procesor.

3.5.3.1. Mașina cu trei adrese

Modelul mașinii cu trei adrese conține în memorie atât **datele** cu care lucrează, cât și **instrucțiunile** care descriu programul de executat.

Zonele de date le vom identifica prin numele lor. *Presupunem că fiecare dată încapă pe un cuvânt.*

Formatul instrucțiunilor.

O instrucțiune la mașina cu trei adrese ocupă patru cuvinte:

cod	adr1	adr2	adr3
-----	------	------	------

Semnificația cuvintelor este:

- ✓ cod – indică codul operației de efectuat;
- ✓ adr1, adr2, adr3 – indică trei locații din memoria mașinii.

O astfel de instrucțiune se execută prin:

$$\text{adr3} := (\text{adr1}) \text{ cod } (\text{adr2})$$

adică, între numărul conținut în locația de memorie indicată de adr1 (ca prim operand) și numărul conținut în locația indicată de adr2 (ca al doilea operand) se efectuează operația indicată de cod. Numărul obținut ca rezultat este depus în locația indicată de adr3.

Instrucțiunile se execută în secvență, una după alta. Schimbarea ordinii de execuție a instrucțiunilor se face cu ajutorul instrucțiunilor de salt.

Pentru a ne referi mai ușor la ele, unele instrucțiuni pot fi identificate printr-un nume:

nume:	cod	adr1	adr2	adr3
-------	-----	------	------	------

Instrucțiunile mașinii cu trei adrese.

Mașina cu trei adrese lucrează cu trei tipuri de instrucțiuni:

1. instrucțiuni de calcul;
2. instrucțiuni de salt;
3. instrucțiuni speciale.

În continuare notăm adresele de memorie cu litere mari (A, B, C etc.), atât pentru locațiile din zona de date cât și pentru cele din zona de instrucțiuni.

1. Mașina cu trei adrese recunoaște 7 **instrucțiuni de calcul** și anume, patru instrucțiuni pentru operațiile aritmetice fundamentale și trei instrucțiuni pentru operații unare uzuale:

+	A	B	C	Adunare	$C := A + B$
-	A	B	C	Scădere	$C := A - B$
*	A	B	C	Inmulțire	$C := A * B$
/	A	B	C	Impărțire	$C := A / B$
$\sqrt{\quad}$	A	-	C	Radical de ord 2	$C := \sqrt{A}$
	A	-	C	Valoare absolută	$C := A $
[]	A	-	C	Partea întreagă	$C := [A]$

2. **Instrucțiunile de salt** sunt de două tipuri: de salt condiționat și de salt necondiționat.

Mașina cu trei adrese recunoaște o singură *instrucțiune de salt necondiționat*:

SN	-	-	A	Următoarea instrucțiune care se va executa este cea de la adresa A
----	---	---	---	--

O *instrucțiune de salt condiționat* comandă saltul la instrucțiunea de la adresa adr3 numai dacă numerele de la adresele adr1 și adr2 se află în relația de ordine specifică instrucțiunii respective. Mașina cu trei adrese recunoaște 6 instrucțiuni de salt condiționat:

<	A	B	C	Salt la instrucțiunea indicată prin C dacă $A < B$
\leq	A	B	C	Salt la instrucțiunea indicată prin C dacă $A \leq B$
>	A	B	C	Salt la instrucțiunea indicată prin C dacă $A > B$
\geq	A	B	C	Salt la instrucțiunea indicată prin C dacă $A \geq B$
=	A	B	C	Salt la instrucțiunea indicată prin C dacă $A = B$
\neq	A	B	C	Salt la instrucțiunea indicată prin C dacă $A \neq B$

3. **Instrucțiunile speciale** ale mașinii cu trei adrese sunt: CIT, TIP și STOP.

CIT	-	-	A	Citește de la intrare valoarea unui număr și o depune în locația indicată prin adresa A
TIP	-	-	A	Afișează la ieșire valoarea conținută în locația indicată prin adresa A
STOP	-	-	-	Incheie execuția programului (oprire)

3.5.3.2. Mașina cu o adresă

Arhitectura mașinii cu o adresă este ceva mai complexă decât a modelului prezentat anterior, astfel că se apropie mai mult de construcția reală a unui procesor.

A. Structura memoriei la mașina cu o adresă.

Principalele componente ale memoriei mașinii cu o adresă sunt:

1. *registrii generali*;
2. zona de memorie rezervată pentru *program*;
3. zona de memorie rezervată pentru *date*;
4. zona de memorie rezervată pentru *stivă*.

1. **Registrii** mașinii cu o adresă sunt trei:

1.	registru acumulator, A	<ul style="list-style-type: none"> – are capacitatea de un cuvânt; – reține la un moment dat unul dintre operanzii operației ce urmează a fi executată și după execuție reține rezultatul operației;
2.	registru index, I	<ul style="list-style-type: none"> – are capacitatea de un cuvânt; – este folosit pentru a accesa componentele unui vector de numere memorat în zona de date;
3.	registru de bază, B	<ul style="list-style-type: none"> – este utilizat pentru a se permite comunicarea între diferite părți ale unui program.

Observație. Depunerea unui număr într-un registru face ca vechiul număr din registru să se piardă. În același timp, conținutul unui registru nu se pierde până când nu se depune un nou număr.

2. **Zona de program** conține instrucțiunile care urmează a fi executate.

O instrucțiune se reprezintă pe două cuvinte și are formatul:

nume:	cod	adr
-------	-----	-----

Semnificația câmpurilor din format este dată de:

$$A := (A) \text{ cod } (\text{adr})$$

adică, se execută operația indicată de cod folosind conținutul curent al registrului acumulator, A și conținutul locației de la adresa adr.

3. **Zona de date** este zona în care sunt memorate valorile variabilelor cu care lucrează programul. Fiecare variabilă simplă se reprezintă pe un cuvânt. Pe lângă variabilele simple, la această mașină se poate opera ușor și cu variabile cu indici: vectori, matrice bidimensionale, tablouri tridimensionale etc. Acestea din urmă vor fi memorate element cu element în locații succesive.

Observație. Depunerea unui număr într-o locație face ca vechiul număr memorat să se piardă. În același timp, conținutul nu se pierde până când nu se depune un nou număr.

4. **Stiva** este o zonă de memorie pentru care accesul la locații este gestionat de disciplina LIFO și de următoarele reguli:

- ✓ în fiecare moment se depune sau se extrage un singur cuvânt din stivă;
- ✓ nu se poate extrage un număr din stivă decât dacă înainte au fost deja depuse numere în stivă;

- ✓ la extragerea unui număr din stivă numărul respectiv se șterge de pe stivă;
- ✓ numerele neextrase din stivă își păstrează valorile.

Observație. Tentativele de efectuare a unei operații de depunere într-o stivă plină sau de extragere dintr-o stivă vidă (situațiile de *depășire*) conduc la oprirea mașinii cu o adresă.

B. Instrucțiunile mașinii cu o adresă.

Pentru prezentarea instrucțiunilor mașinii cu o adresă convenim să notăm cu **X** o adresă oarecare din zona de date, sau o adresă din zona de program, sau o constantă.

Mașina cu o adresă recunoaște 28 instrucțiuni considerate în patru grupe:

instrucțiuni pentru transferuri de date;

- a) instrucțiuni de calcul;
- b) instrucțiuni de salt;
- c) instrucțiuni speciale.

Pentru mașina cu o adresă putem identifica o instrucțiune în mod unic și printr-un număr de la 1 la 28.

În tabelele următoare, pentru fiecare instrucțiune precizăm: numărul de ordine, codul, adresa locației pe care se execută instrucțiunea și, eventual, comentarii specifice fiecărei instrucțiuni.

a) Instrucțiunile pentru transferuri de date sunt:

1	←	X	Valoarea indicată de X se depune în registrul acumulator A
2	→	X	Valoarea din registrul acumulator A se depune în locația din zona de date indicată de X
3	← _I		Conținutul registrului index I se depune în registrul acumulator A
4	→ _I		Conținutul registrului acumulator A se depune în registrul index I
5	← _B		Conținutul registrului de bază B se depune în registrul acumulator A
6	→ _B		Conținutul registrului acumulator A se depune în registrul de bază B
7	↓	X	Valoarea indicată prin X este depusă în vârful stivei; numărul elementelor din stivă crește cu 1
8	↑	X	Este extras elementul din vârful stivei și este depus în locația de date indicată de X; numărul elementelor din stivă scade cu 1
9	↓		Valoarea din registrul acumulator A este depusă în vârful stivei; numărul elementelor din stivă crește cu 1
10	↑		Este extras elementul din vârful stivei și este depus în registrul acumulator A; numărul elementelor din stivă scade cu 1

b) Instrucțiunile de calcul coincid cu cele recunoscute de mașina cu trei adrese:

11	+	X	Adunare	$A := A + X$
12	-	X	Scădere	$A := A - X$
13	*	X	Inmulțire	$A := A * X$
14	/	X	Împărțire	$A := A / X$
15	√	X	Radical de ord 2	$A := \sqrt{X}$

16		X	Valoare absolută $A:= X $
17	[]	X	Partea întreagă $A:=[X]$

c) **Instrucțiunile de salt** pot fi condiționate sau nu. Si mașina cu o adresă recunoaște o singură *instrucțiune de salt necondiționat*:

18	SN	X	Următoarea instrucțiune care se va executa este cea de la adresa din zona de program indicată de X
----	----	---	--

Instrucțiunile de salt condiționat determină un salt la instrucțiunea indicată de X dacă conținutul acumulatorului A este într-o anumită relație de ordine față de zero. Avem următoarele 6 instrucțiuni de salt condiționat:

19	<	X	Salt la X dacă $A < 0$
20	\leq	X	Salt la X dacă $A \leq 0$
21	>	X	Salt la X dacă $A > 0$
22	\geq	X	Salt la X dacă $A \geq 0$
23	=	X	Salt la X dacă $A = 0$
24	\neq	X	Salt la X dacă $A \neq 0$

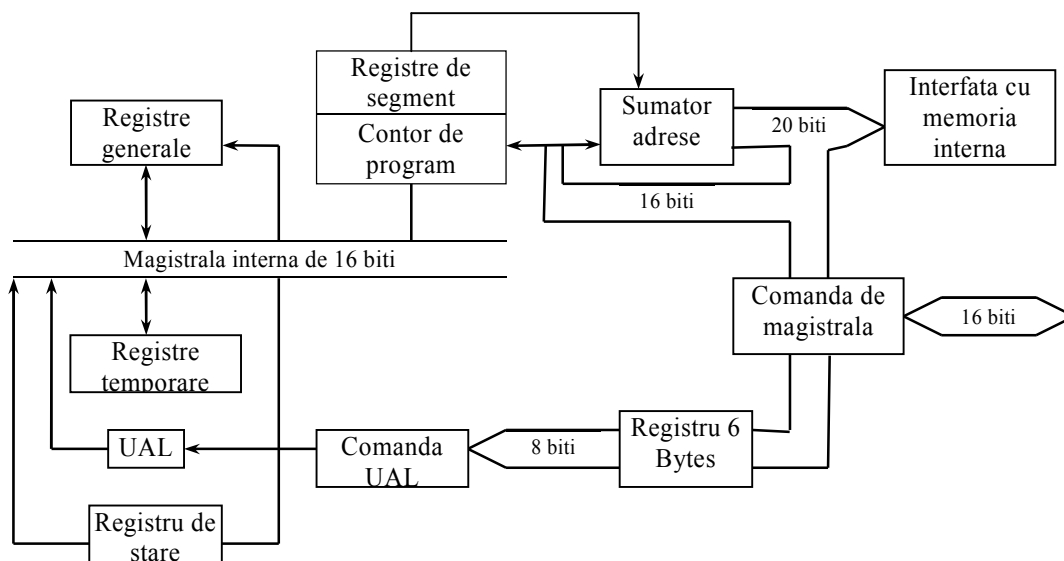
d) **Instrucțiunile speciale** ale mașinii cu o adresă completează cu o instrucțiune setul de la mașina cu trei adrese:

25	ADR	X	Determină <u>adresa</u> (nu valoarea) din zona de date a variabilei indicate de X și depune această adresă în registrul acumulator A
26	CIT	X	Citește de la intrare valoarea unui număr și o depune în locația din zona de date indicată de X
27	TIP	X	Afișează la ieșire valoarea numărului aflat în locația din zona de date indicată de X
28	STOP		Incheie execuția programului (oprire)

NOTIȚE

3.5.4. Structura internă de bază a unui microprocesor I 80x86

Complicând modelele din paragraful anterior, prezentăm aici schema unui microprocesor din familia Intel 80x86 pe 16 biți.



Registrele generale sunt:

a) **registrele de utilizare generală** se notează cu:

AX - registrul acumulator general

BX - registrul de bază

CX - registrul numărător

DX - registrul de date.

Acestea pot fi adresate în două moduri: direct pe 16 biți sau adresând separat byte-ul superior (primii 8 biți) și byte-ul inferior (următorii 8 biți). Corespunzător, cuvintele se vor numi:

15	8	7	0
AH		AL	
BH		BL	
CH		CL	
DH		DL	

b) **registrul indicator de stivă** de 16 biți. Acesta conține adresa vârfului stivei, SP (*stack pointer*).

c) **registrul indicator de bază** de 16 biți. Acesta conține adresa de bază, BP (*base pointer*).

d) **registrele de index**: SI (*source index*) și DI (*destination index*), de 16 biți fiecare. Aceștia participă la elaborarea adreselor, fiecare adresă rezultând prin însumarea diferitelor combinații între adresa de bază, un indice relativ la poziția curentă și o deplasare (numită *adresă de offset*).

Registrul de stare este registrul indicatorilor de condiție. Fiecare subrutină care folosește registrul de flaguri va interpreta de o manieră proprie valorile găsite. Cel mai des, indicatorii de condiție sunt folosiți în instrucțiunile de salt.

Cei 16 indicatori de condiție ai registrului de flag-uri au următoarea semnificație:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
X	X	X	X	OF	DF	IF	TF	SF	ZF	X	AF	X	PF	X	CF

X - rezervați;

CF - indicator de transport din MSB al rezultatului;

PF - indicator de paritate;

AF - indicator de transport de la bitul 4 la bitul 5;

ZF - indicator de rezultat nul;

SF - indicator de semn negativ;

TF - indicator de întrerupere internă;

IF - indicator de întrerupere externă permisă;

DF - indicator de decrementare / incrementare a regiștrilor SI sau DI după execuția unei operații;

OF - indicator de depășire aritmetică.

Registrele de segment ale microprocesorului permit adresarea memoriei interne, fiecare având o capacitate de 16 biți. Aceștia sunt:

CS - Cod Segment - segmentul pentru program (cod sursă);

DS - Data Segment - segmentul pentru date;

SS - Stack Segment - segmentul pentru stivă;

ES - Extra Segment - segmentul pentru date auxiliare.

Contorul de program, PC sau **IP** (*Instruction Pointer*) are o capacitate de 16 biți și conține o adresă relativă (*offset*) a instrucțiunii curente din segmentul de program CS.

De exemplu, pentru o instrucțiune de salt, IP este salvat în vârful stivei și încărcat cu adresa relativă în segmentul program.

Registru de 6 bytes conține instrucțiuni și are rolul de a îmbunătăți viteza de prelucrare prin două mecanisme distincte:

datele preluate din memorie sunt plasate în acest registru în ordinea introducerii; astfel, UAL poate să preia instrucțiunile fără să fie influențată de timpul de acces al memoriei;

la transferul operanzilor între memorie sau dispozitive periferice și UAL se generează și adresele necesare.

Rezumat.

În prezentarea structurii organizatorice și funcționale a unui sistem de calcul un loc aparte trebuie rezervat microprocesorului – creierul calculatorului și reprezentarea fizică cea mai importantă a UCP.

În prima parte a acestui paragraf este descris procesorul din diferite puncte de vedere: istoric, caracteristici generale, caracteristici de performanță. Apoi sunt tratate aspecte concrete legate de cele trei tipuri de arhitecturi de procesoare existente pe piață: CISC, RISC și MIPS.

În pregătirea capitolului al patrulea, paragrafele 3.8.3 și 3.8.4 prezintă trei *nivele* de structuri de unități de prelucrare microprogramate. Este vorba de modelul mașinii cu trei adrese, modelul mașinii cu o adresă și, în final, modelul larg răspândit al procesoarelor din familia Intel. Aceste trei exemple reprezintă totodată trei nivele de abstractizare (de la simplu la complex) a

structurii interne a unității de prelucrare. Cel mai simplu model este cel al mașinii cu trei adrese pentru că toate instrucțiunile sunt în mod de adresare directă. La mașina cu o adresă, deja modelul este mai complicat pentru că multe dintre instrucțiunile recunoscute de această mașină lucrează cu regiștrii sau cu alte zone de memorie implicite.

Cuvinte cheie.

procesor
lungimea cuvântului, frecvența de tact
instrucțiune, cod de instrucțiune, setul de bază de instrucțiuni
arhitectură CISC, RISC, MIPS

Verificare.

1. Dacă un procesor are frecvența de tact de 6MHz aceasta înseamnă că poate executa 6 milioane de operații într-o secundă. Dacă considerăm că o astfel de operație este o etapă din *pipeline*-ul intern al procesorului respectiv, să se calculeze viteza de prelucrare a acestui procesor (în Mips). Se va considera că banda de asamblare este cu cinci faze.

3.6. Multiprocesare

Dacă două sau mai multe calculatoare sau procesoare cooperează într-o manieră oarecare atunci totalitatea resurselor lor formează un **sistem distribuit**. De aici rezultă imediat două tipuri de sisteme distribuite: sistemele **multicalculator** și sistemele **multiprocesor**.

Există numeroase asemănări între sistemele multiprocesor și sistemele multicalculator. Cele mai multe asemănări rezultă din faptul că ambele tipuri suportă operații efectuate paralel și / sau concurrent.

Totuși, există o deosebire importantă între un sistem cu mai multe procesoare și un sistem cu mai multe calculatoare. Într-un sistem multicalculator, calculatoarele sunt interconectate între ele prin linii de comunicare formând o **rețea de calculatoare**. Într-o rețea avem mai multe calculatoare autonome (independente) care pot sau nu să comunice fiecare cu fiecare. Comparativ, un sistem multiprocesor este controlat de un același sistem de operare, care asigură interconexiunea între procesoare și toate componentele care concură la realizarea sarcinilor.

Cu alte cuvinte, un sistem multiprocesor rezultă din interconectarea a două sau mai multor procesoare cu memorii și echipament de intrare / ieșire.

Din această definiție trebuie să înțelegem că un procesor poate reprezenta o unitate UCP sau poate fi un procesor IOP. Totuși, calitatea de sistem multiprocesor apare când cel puțin unul dintre procesoare are proprietăți de UCP și nu numai de comunicare de tip IOP.

Sistemele multiprocesor fac parte din categoria MIMD.

Un exemplu imediat de sistem multiprocesor este microprocesorul însuși, cel puțin în ceea ce privește integrarea coprocesorului matematic și dezvoltarea procesoarelor *multi-core*.

Pentru a spori viteza de execuție a operațiilor matematice, arhitectura internă a microprocesoarelor s-a extins cu o unitate specializată în operații cu numere reale exprimate în virgulă mobilă, numită **coprocesor matematic**. Pentru utilizator, această unitate auxiliară este integrată în microprocesorul părinte prin interfața coprocesorului. Cele două componente sunt conectate în paralel, astfel că fiecare poate să-și execute la un moment dat propriile sale instrucțiuni.

Coprocesoarele matematice au urmat evoluția microprocesoarelor. Până la microprocesorul I80486SX coprocesorul este un *chip* distinct de microprocesor și preia numele acestuia, schimbând doar ultima cifră. Astfel, coprocesorul matematic atașat microprocesorului I80x86 se va numi I 80x87. După varianta I80486SX, microprocesoarele I80486DX, I80486DX2, I80486DX4 și Pentium au inclus coprocesorul matematic pe același *chip* cu microprocesorul.

În absența coprocesoarelor, programele folosesc *emulatoare de coprocesoare* care sunt conectate în locul microprocesoarelor preluând astfel controlul întregului sistem.

Rețelele locale sunt apreciate ca cele mai adecvate și răspândite suporturi fizice pentru sistemele cu prelucrare distribuită.

Multiprocesarea crește fiabilitatea sistemului prin faptul că o eroare apărută într-o *parte* are un efect limitat și nu influențează întregul sistem. Mai mult, dacă un procesor *cade*, procesele lui pot fi dirijate spre execuție către un alt procesor.

Un avantaj fundamental al sistemelor multiprocesor constă în posibilitatea procesării paralele a task-urilor în două moduri:

mai multe sarcini independente pot fi executate în paralel sau

o aceeași sarcină poate fi împărțită în mai multe task-uri de executat în paralel.

Rolul acestui paragraf este de a deschide noi orizonturi pentru studentul care și-a însușit noțiunile elementare de arhitectura calculatoarelor prezentate în acest curs și care este astfel în măsură să înțeleagă structuri mai complexe de organizare și realizare a conexiunilor inter-calculatoare.

Multiprogramarea reprezintă modul de exploatare a unui sistem de calcul cu un singur procesor central, dar care presupune existența simultană în memoria internă a mai multor

programe care se execută concurrent (processe concurente); astfel rezultă o mai bună utilizare a UCP și a memoriei.

Un **task** este unitatea elementară delimitată într-o secvență de acțiuni specifice pentru atingerea unui anumit scop. Termenul de **multitasking** sau **multiprelucrare** se referă la capacitatea unui calculator de a executa mai multe taskuri simultan. La o prezentare mai detaliată a conceptului, caracteristica de multitasking se atribuie direct sistemului de operare. Astfel, politica de alocare a taskurilor la unitățile de prelucrare disponibile în sistem ține de organizarea și gestionarea resurselor de către sistemul de operare gazdă. Un task în execuție este un **proces**. Dacă, la un moment dat, sunt executate mai multe procese simultan atunci aceste sunt **processe paralele**. Dacă, în plus, aceste procese evoluează cu interschimb de informații între ele, atunci ele sunt **processe concurente**. Din alt punct de vedere, dacă sistemul gazdă dispune de mai multe unități de prelucrare pentru a executa procesele paralele atunci **paralelismul este efectiv**. Altfel, **paralelismul este simulat** prin diferite politici de alocare proces-procesor.

În ceea ce privește aspectele de paralelism la nivelul structurii interne a sistemului de calcul, se poate spune că în arhitectura calculatoarelor Mark I și ENIAC intrau în calcul la un moment dat mai multe elemente de calcul ce lucrau în paralel la o problemă comună, fiind dirijate de o singură unitate de comandă. La versiunile următoare s-a renunțat la această arhitectură paralelă. Ulterior, în arhitectura procesoarelor Intel s-a revenit la paralelizarea etapelor de prelucrare. Astfel, dacă la 8086 erau doar două unități funcționale care lucrau în paralel (EU și BIU), structura procesorului 286 conține 4 unități funcționale paralele, în timp ce procesorul 386 conține 6 astfel de unități funcționale și anume:

1. unitatea de interfață cu magistrala (BIU, engl. *Bus Interface Unit*);
2. unitatea de citire în avans a instrucțiunilor (engl. *Code Prefetch Unit*);
3. unitatea de decodificare a instrucțiunii (engl. *Instruction Decode Unit*);
4. unitatea de execuție (EU, engl. *Execution Unit*);
5. unitatea de traducere a adreselor (engl. *Segment Unit*);
6. unitatea de paginare (engl. *Paging Unit*).

Mai multe detalii despre paralelismul intern la nivelul procesorului se pot consulta în [Lungu, 2000].

Rezumat.

Acest paragraf a încercat să atragă atenția studenților că *arhitectura calculatoarelor* este o poartă care deschide reale perspective de documentare și însușire de cunoștințe în domenii moderne ca: rețele de calculatoare, sisteme distribuite, prelucrare nesecvențială (paralelă și/sau concurrentă). Conceptele nu sunt tratate aici exhaustiv, fiecare dintre ele reprezentând în fond obiectul unei alte discipline de studiu.

Verificare.

1. Care sunt avantajele prelucrării în paralel a sarcinilor de calcul și/sau de control?
2. Care sunt principalele unități funcționale ale calculatorului care lucrează în paralel?
3. Comparați structura benzii de asamblare a procesorului în cinci stagii (paragraful *Moduri de adresare*) cu funcțiile celor șase unități interne de prelucrare ale procesorului descrise în finalul paragrafului.

NOTIȚE

ARHITECTURA CALCULATOARELOR

Lector Dr. Anca Vasilescu

PARTEA a-II-a

2007 - 2008



CUVÂNT ÎNAINTE



Materialul didactic scris pentru studenți este împărțit în: (I) suport pentru activitățile tutoriale – care conține considerentele teoretice corespunzătoare capitolelor 1 și 3 din tematica cursului și (II) suport pentru activitățile aplicative – care conține considerentele teoretice și problemele rezolvate corespunzătoare capitolului 2 din tematica cursului.



TEMATICA CURSULUI

⇒ Componentele de bază ale unui sistem de calcul

- 🔗 Noțiuni introductive
 - ✍ Definiții. SC – din diferite puncte de vedere. Structura unui sistem de calcul. Performanțe. Clasificări. Principiile von Neumann. Modelul formal al unui calculator.
- 🔗 Sistemul de intrare/ ieșire. Dispozitive periferice.
 - ✍ Clasificări. Canalul de intrare/ ieșire. Legarea perifericelor la SC. Procesoare de intrare/ ieșire. Structura unor clase de periferice.
- 🔗 Unitatea de memorie
 - ✍ Definiții. Structura fizică. Structura ierarhică de organizare a memoriei. Tipuri de acces la memorii.

⇒ Abordarea la nivel logic-digital a componentelor calculatorului

- 🔗 Circuite logice
 - ✍ Porți logice. CLC. CLS. CLC aritmetice. CLC decodoare. CLC multiplexoare. CLS bistabile. Circuite integrate.
- 🔗 Regiștri
 - ✍ Regiștri cu încărcare paralelă. Regiștri de deplasare. Numărătoare binare. Regiștri de incrementare.
- 🔗 Structura de bază a unui calculator
 - ✍ Regiștri de bază ai unui sistem de calcul. Unitatea de contorizare și control. Transferul datelor pe magistrală.

⇒ Unitatea centrală de prelucrare

- 🔗 Organizarea generală a regiștrilor. Organizarea stivei
- 🔗 Microoperație. Instrucțiuni. Program. Codurile instrucțiunilor
- 🔗 Moduri de adresare
- 🔗 Clasificarea instrucțiunilor
 - ✍ Instrucțiuni pentru transferul datelor. Instrucțiuni pentru prelucrarea datelor. Instrucțiuni pentru controlul programului.
- 🔗 Prelucrarea instrucțiunilor unui program
- 🔗 Microprocesorul
 - ✍ Caracteristici. Setul de instrucțiuni. Modelarea funcționării unui procesor. Mașina cu trei adrese. Mașina cu o adresă. Structura internă de bază a unui microprocesor I 80x86.
 - ✍ Introducere în utilizarea limbajului de asamblare. Sintaxa principalelor instrucțiuni. Rezolvarea unor probleme elementare.
- 🔗 Multiprocesare. Coprocesoare matematice.



ABORDAREA LA NIVEL LOGIC-DIGITAL A COMPONENTELOR UNUI CALCULATOR

2.1.	Circuite logice -----	4
2.1.1	Porți logice. CLC. CLS -----	4
2.1.2	CLC aritmetice-----	6
2.1.2.1	CLC sumator -----	7
2.1.2.2	CLC scăzător -----	10
2.1.2.3	CLC sumator cu comutare -----	12
2.1.3	CLC decodare -----	14
2.1.4	CLC multiplexoare -----	15
2.1.5	CLS bistabile -----	17
2.1.6	Circuite integrate -----	22
2.2.	Registre -----	25
2.2.1	Registre de microoperație -----	25
2.2.1.1	Registru cu încărcare paralelă -----	25
2.2.1.2	Registru de deplasare -----	26
2.2.1.3	Numărătoare binare -----	28
2.2.1.4	Registru de incrementare -----	29
2.2.2	Transferul datelor din regiștri pe magistrală -----	30

Un sistem de calcul modern este un sistem digital, un ansamblu de componente digitale care concură la prelucrarea datelor și la executarea instrucțiunilor primite. Aceste componente constau din circuite logice, grupate în componente aritmetice, decodare, regiștrii ș.a. Conectate la magistralele sistemului, aceste componente interacționează și astfel, îndeplinesc misiunea primită de la utilizator.

În capitolul de față vom descrie aceste componente interne al nivel logic-digital.

3.1. Circuite logice

3.1.1 Porți logice. CLC. CLS

Informația binară este reprezentată în sistemul de calcul digital prin cantități fizice, numite **semnale**, care se transmit conform cu construcția internă a fiecărei componente. Cele mai simple semnale electrice sunt de tip on / off.

Pentru descrierea funcționării calculatorului este util să construim pentru componentele de bază traseul semnalelor electrice, adică să prezentăm componentele respective prin **circuitele logice** interne.

Semnalele electrice sunt date de anumite tensiuni și se regăsesc în sistem în una din două stări posibile. Aceste stări corespund valorilor unei variabile binare, adică 0 sau 1. De exemplu, un SC particular poate lucra cu o tensiune de 3V pentru starea corespunzătoare valorii 1 și o tensiune de 0.5V pentru starea corespunzătoare valorii 0. În acest caz, intrările în CLC recunosc tensiuni de 0.5V și 3V, iar la ieșire circuitul *răspunde* cu o tensiune egală tot cu una dintre aceste valori.

Pentru buna înțelegere a celor ce urmează este util să introducem aici câteva noțiuni.

Dacă notăm $B_2 = \{0,1\}$ și $B_2^n = B_2 \times B_2 \times \dots \times B_2$ atunci sunt imediate următoarele definiții:

Definiția 1. Se numește **funcție logică** o aplicație $f: B_2^n \rightarrow B_2^m$.

Definiția 2. Dacă $m = 1$ atunci **funcția logică** se numește **scalară**. Altfel, spunem că avem o **funcție logică vectorială**.

Reprezentarea unei funcții logice se poate da prin tabelă de valori (tabelă de adevăr), prin expresie logică sau prin circuitul logic asociat.

În practică se remarcă unele funcții, numite **funcții logice elementare**: **ȘI LOGIC** (conjuncția), **SAU LOGIC** (disjuncția), **NU LOGIC** (negarea sau complementarea). Circuitele logice asociate acestor funcții se numesc **porți logice elementare**.

Convenim să notăm conjuncția multiplicativ, disjuncția aditiv și negarea prin operatorul *apostrof*.




Tabelele de valori pentru aceste funcții sunt:

a	b	a b
0	0	0
0	1	0
1	0	0
1	1	1

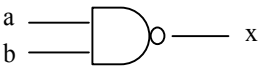
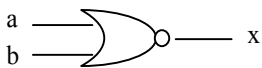


a	b	a+b
0	0	0
0	1	1
1	0	1
1	1	1

a	a'
0	1
1	0

Caracterizarea funcțiilor logice elementare este dată în tabelul următor:

<i>Nume</i>	<i>Simbol grafic</i>	<i>Expresie logică</i>
ȘI LOGIC		$x = a \cdot b$ sau $x = a \wedge b$
SAU LOGIC		$x = a + b$
NU LOGIC		$x = a'$

Pe lângă porțile logice elementare, reprezentarea cu circuite apelează la porțile logice ale unor funcții uzuale. Acestea sunt: **NAND**, **NOR**, **XOR** și **NXOR**, definite și caracterizate conform cu următorul tabel:

<i>Nume</i>	<i>Simbol grafic</i>	<i>Expresie logică</i>
NAND și – nu		$x = (a \cdot b)'$
NOR sau – nu		$x = (a + b)'$
XOR sau exclusiv (anticoincidență)		$x = a'b + ab' =$ $= a \oplus b$
NXOR NOR exclusiv (echivalență) (coincidență)		$x = ab + a'b' =$ $= (a \oplus b)'$

Funcțiile uzuale introduse anterior sunt definite de următoarele tabele de valori:

a	b	$(a \cdot b)'$	a	b	$(a+b)'$	a	b	$a \oplus b$	a	b	$(a \oplus b)'$
0	0	1	0	0	1	0	0	0	0	0	1
0	1	1	0	1	0	0	1	1	0	1	0
1	0	1	1	0	0	1	0	1	1	0	0
1	1	0	1	1	0	1	1	0	1	1	1

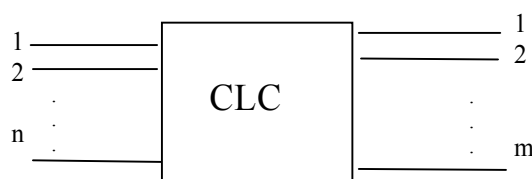
Pentru simplificarea funcțiilor logice trebuie cunoscute un minim de identități din algebra booleană:

(1) $a + 0 = a$	(1) $a \cdot 0 = 0$
(2) $a + 1 = 1$	(2) $a \cdot 1 = a$
(3) $a + a = a$	(3) $a \cdot a = a$
(4) $a + a' = 1$	(4) $a \cdot a' = 0$
(5) $a + b = b + a$	(5) $ab = ba$
(6) $a + (b + c) = (a + b) + c$	(6) $a(bc) = (ab)c$
(7) $a(b + c) = ab + ac$	(7) $a + bc = (a + b)(a + c)$
(8) $(a + b)' = a'b'$	(8) $(ab)' = a' + b'$
(9) $(a')' = a$	

Definiția 3. Dacă o funcție este reprezentată prin circuit logic atunci argumentele funcției sunt **intrări în circuit**, iar valorile funcției, pentru fiecare combinație a valorilor de intrare, reprezintă **ieșirile din circuit**.

Definiția 4. Un **circuit logic combinațional (CLC)** este un circuit logic în care valorile de ieșire depind numai de valorile de intrare în circuit. Altfel, dacă ieșirile depind și de intrări și de stările intermediare ale componentelor interne ale circuitului, atunci avem un **circuit logic secvențial (CLS)**.

Prin convenție, un circuit logic combinațional cu n intrări și m ieșiri se reprezintă astfel:



În paragrafele următoare vom detalia analiza unor circuite logice care se remarcă prin importanța pe care o au în sistemele de calcul.

3.1.2 CLC aritmetice

Pentru efectuarea operațiilor aritmetice sistemul de calcul este prevăzut cu componente specializate care alcătuiesc unitatea aritmetico-logică, UAL. Structura internă a acestor componente este dată de circuite logice combinaționale construite astfel încât să determine la ieșire rezultatul operării valorilor de intrare.

În acest paragraf ne referim la componentele UAL pentru efectuarea operațiilor aritmetice.

3.1.2.1 CLC sumator

Fie două numere naturale A și B date în reprezentare binară pe n poziții prin:

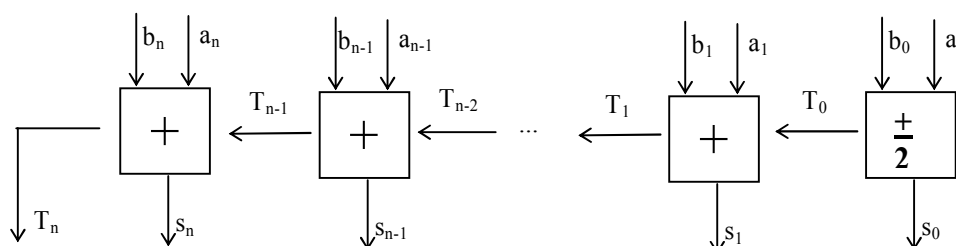
$$A = a_n a_{n-1} \dots a_2 a_1 a_0 \text{ și}$$

$$B = b_n b_{n-1} \dots b_2 b_1 b_0.$$

Observație. Dacă cifrele semnificative ale unuia dintre numere nu ocupă toate cele n poziții atunci acestea se încarcă prin aliniere la dreapta și zona neocupată se încarcă cu zerouri.

Notăm cu S suma celor două numere A și B. Reprezentarea binară a lui S este $S = s_n s_{n-1} \dots s_2 s_1 s_0$.

Scopul acestui paragraf este să construim un circuit logic combinațional care să aibă ca intrări valorile cifrelor binare ale operanzilor și ca ieșiri cifrele binare ale sumei, numit **CLC sumator total**. Schematic, acesta se reprezintă astfel:



Am notat cu T_i , $i = 0, 1, \dots, n-1, n$ cifra de transport de la rangul i la rangul $i + 1$. Deoarece cifra de transport de la ultimul rang, T_n , face parte din rezultat, ieșirea aceasta s-a legat la masă. În consecință, ea va fi citită ca și cifră rezultat împreună cu cifrele s_0, s_1, \dots, s_n ale sumei S.

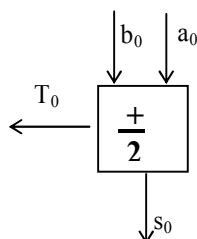
Ținând cont de ordinea în care se execută operațiile aritmetice, circuitul sumator reprezentat în figura anterioară este un **sumator secvențial cu încărcare paralelă**.

În reprezentarea de mai sus se disting două tipuri de circuite, și anume:

- circuit de adunare cu două intrări și două ieșiri, numit **circuit semisumator** și
- circuit de adunare cu trei intrări și două ieșiri, numit **circuit sumator complet**.

Avem circuit semisumator pentru operare pe rangul zero, în care se adună cifrele operanzilor și rezultă cifra sumei și cifra de transport. În rest, pentru operarea pe celelalte ranguri, avem circuite sumatoare complete, în care se adună cifrele operanzilor cu cifra de transport de la rangul anterior și rezultă cifra sumei și cifra de transport la rangul următor.

a) Circuitul semisumator se reprezintă schematic astfel:



Fiecare ieșire corespunde unei funcții logice. Pentru descrierea funcționării circuitului vom determina mai întâi expresiile logice ale acestor funcții.

Tabla adunării în baza 2 determină următorul tabel de valori pentru funcțiile s_0 și T_0 :

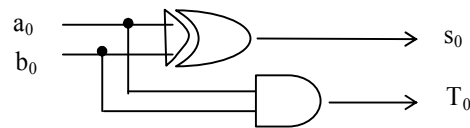
a	b	a+b		a ₀	b ₀	s ₀	T ₀
0	0	0	=>	0	0	0	0
0	1	1		0	1	1	0
1	0	1		1	0	1	0
1	1	10		1	1	0	1

De aici putem deduce următoarele expresii logice:

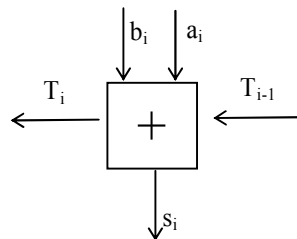
$$s_0 = a_0'b_0 + a_0b_0' = a_0 \oplus b_0 \text{ și}$$

$$T_0 = a_0b_0.$$

Corespunzător, circuitul logic semisumator se reprezintă prin următoarele porți logice:



b) Circuitul sumator complet pentru adunarea cifrelor binare de rang i se reprezintă schematic astfel:



Pentru descrierea funcționării circuitului vom proceda ca și în cazul circuitului semisumator. Pentru început, determinăm expresiile logice ale funcțiilor s_i și T_i de ieșire.

Avem următorul tabel de valori pentru funcțiile s_i și T_i :

a _i	b _i	T _{i-1}	s _i	T _i
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

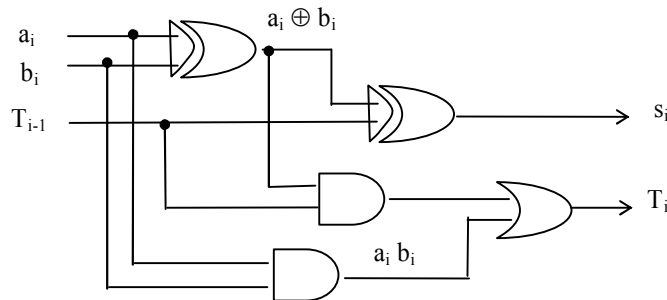
De aici putem deduce următoarele expresii logice:

$$\begin{aligned}
 s_i &= a_i'b_i'T_{i-1} + a_i'b_iT_{i-1}' + a_ib_i'T_{i-1}' + a_ib_iT_{i-1} = \\
 &= (a_i'b_i' + a_ib_i) T_{i-1} + (a_i'b_i + a_ib_i') T_{i-1}' = \\
 &= (a_i \oplus b_i)' T_{i-1} + (a_i \oplus b_i) T_{i-1}' = \\
 &= T_{i-1} \oplus (a_i \oplus b_i)
 \end{aligned}$$

și

$$\begin{aligned}
T_i &= a_i' b_i T_{i-1} + a_i b_i' T_{i-1} + a_i b_i T_{i-1}' + a_i b_i T_{i-1} = \\
&= (a_i' b_i + a_i b_i') T_{i-1} + a_i b_i (T_{i-1}' + T_{i-1}) = \\
&= (a_i \oplus b_i) T_{i-1} + a_i b_i = \\
&= a_i b_i + T_{i-1} (a_i \oplus b_i).
\end{aligned}$$

Corespunzător, circuitul logic sumator complet se reprezintă prin următoarele porți logice:



Pentru a reprezenta circuitul sumator total mai avem un singur pas, care constă în determinarea unei expresii pentru T_i care să nu fie recursivă.

Notăm

$$P_i := a_i \oplus b_i, (\forall) i = 0, 1, 2, \dots, n \text{ și}$$

$$G_i := a_i b_i, (\forall) i = 0, 1, 2, \dots, n.$$

Cu aceste notații, sunt imediate următoarele relații:

$$s_0 = P_0, T_0 = G_0,$$

și

$$T_1 = G_1 + P_1 T_0 = G_1 + P_1 G_0,$$

$$T_2 = G_2 + P_2 T_1 = G_2 + P_2 G_1 + P_2 P_1 G_0,$$

$$T_3 = G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 G_0.$$

De aici, putem anunța că, pentru orice i , este verificată relația

$$T_i = G_i + P_i G_{i-1} + P_i P_{i-1} G_{i-2} + \dots + P_i P_{i-1} \dots P_1 G_0.$$

Evident, relația este complet justificată după aplicarea unei inducții matematice.

Folosind această relație pentru T_i , suntem acum în măsură să construim circuitul sumator total. Pentru simplitatea reprezentării vom alege $n=3$, adică situația în care fiecare operand are 4 poziții binare.

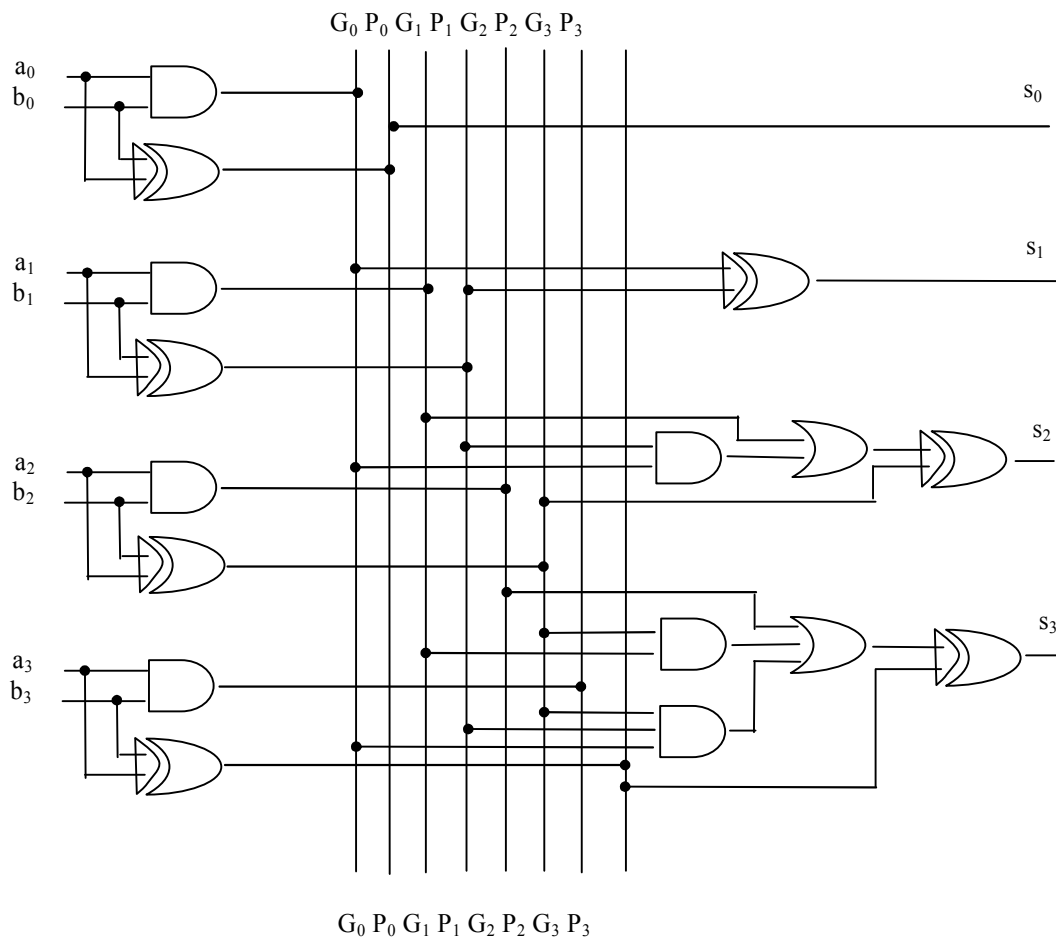
Expresiile logice pentru cifrele sumei sunt:

$$s_0 = P_0;$$

$$s_1 = G_0 \oplus P_1;$$

$$s_2 = (G_1 + P_1 G_0) \oplus P_2;$$

$$s_3 = (G_2 + P_2 G_1 + P_2 P_1 G_0) \oplus P_3.$$



3.1.2.2 CLC scăzător

Urmând pașii construcției din paragraful anterior, vom descrie aici circuitul logic combinațional care determină diferența a două numere naturale.

Fie cele două numere naturale A și B date în reprezentare binară pe n poziții prin:

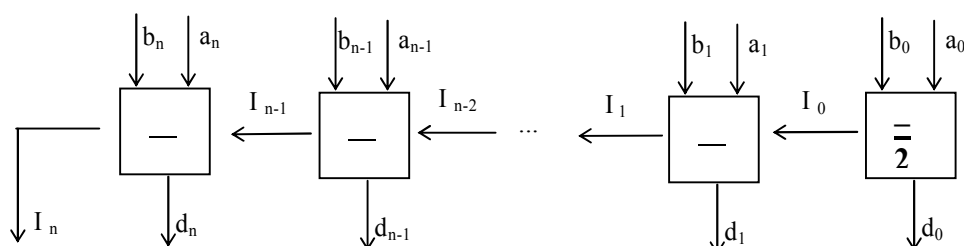
$$A = a_n a_{n-1} \dots a_2 a_1 a_0 \text{ și}$$

$$B = b_n b_{n-1} \dots b_2 b_1 b_0.$$

Notăm cu D diferența celor două numere A și B. Reprezentarea binară a lui D este $D = d_n d_{n-1} \dots d_2 d_1 d_0$.

Circuitul logic combinațional scăzător are ca intrări valorile cifrelor binare ale operandilor și ca ieșiri cifrele binare ale diferenței.

Schematic, acesta se reprezintă astfel:

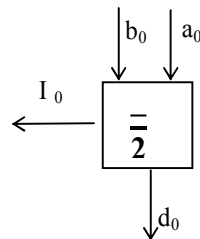


Am notat cu I_i , $i = 0, 1, \dots, n-1$, n cifra de împrumut de la rangul $i+1$ la rangul i . Deoarece cifra de împrumut pentru ultimul rang, I_n , face parte din rezultat, ieșirea aceasta s-a legat la masă. În consecință, ea va fi citită ca și cifră rezultat împreună cu cifrele d_0, d_1, \dots, d_n ale diferenței D .

În reprezentarea de mai sus se disting două tipuri de circuite, și anume:

- circuit de scădere cu două intrări și două ieșiri, numit **circuit semiscăzător** și
- circuit de scădere cu trei intrări și două ieșiri, numit **circuit scăzător complet**, cu precizările introduse la circuitul sumator.

a) **Circuitul semiscăzător** se reprezintă schematic astfel:



Pentru descrierea funcționării circuitului vom determina expresiile ale funcțiilor logice de ieșire, I_0 și d_0 .

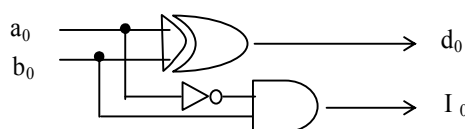
a_0	b_0	d_0	I_0
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

De aici putem deduce următoarele expresii logice:

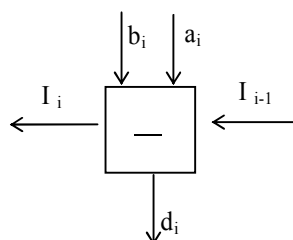
$$d_0 = a_0' b_0 + a_0 b_0' = a_0 \oplus b_0 \text{ și}$$

$$I_0 = a_0' b_0.$$

Corespunzător, circuitul logic semiscăzător se reprezintă prin următoarele porți logice:



b) **Circuitul scăzător complet** pentru scăderea cifrelor binare de rang i se reprezintă schematic astfel:



Expresiile logice ale funcțiilor de ieșire d_i și I_i sunt date de următorul tabel de valori. Calculul constă în $a_i - I_{i-1} - b_i$ și rezultă cifra d_i a diferenței și împrumutul I_i .

a_i	b_i	I_{i-1}	d_i	I_i
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

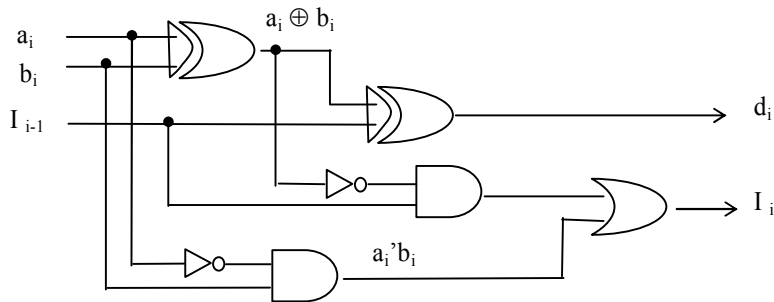
De aici putem deduce următoarele expresii logice:

$$\begin{aligned}
 d_i &= a_i'b_i'I_{i-1} + a_i'b_iI'_{i-1} + a_ib_i'I'_{i-1} + a_ib_iI_{i-1} = \\
 &= (a_i'b_i' + a_ib_i) I_{i-1} + (a_i'b_i + a_ib_i') I'_{i-1} = \\
 &= (a_i \oplus b_i)' I_{i-1} + (a_i \oplus b_i) I'_{i-1} = \\
 &= I_{i-1} \oplus (a_i \oplus b_i)
 \end{aligned}$$

și

$$\begin{aligned}
 I_i &= a_i'b_i'I_{i-1} + a_i'b_iI'_{i-1} + a_i'b_iI_{i-1} + a_ib_iI_{i-1} = \\
 &= (a_i'b_i' + a_ib_i) I_{i-1} + a_i'b_i (I'_{i-1} + I_{i-1}) = \\
 &= (a_i \oplus b_i)' I_{i-1} + a_i'b_i = \\
 &= a_i'b_i + I_{i-1}(a_i \oplus b_i)'.
 \end{aligned}$$

Corespunzător, circuitul logic scăzător complet se reprezintă prin următoarele porți logice:



3.1.2.3 CLC sumator cu comutare

Expresiile logice pentru funcțiile de ieșire din circuitele aritmetice anterioare au fost:

$$s_0 = a_0 \oplus b_0 = d_0$$

$$s_i = T_{i-1} \oplus (a_i \oplus b_i), (\forall) i=1..n$$

$$d_i = I_{i-1} \oplus (a_i \oplus b_i), (\forall) i=1..n$$

și, respectiv,

$$T_0 = a_0b_0$$

$$I_0 = a_0'b_0$$

$$T_i = a_ib_i + T_{i-1}(a_i \oplus b_i), (\forall) i=1..n$$

$$I_i = a_i' b_i + I_{i-1} (a_i \oplus b_i)', (\forall) i = 1..n.$$

Analizând aceste formule și circuitele asociate lor, se observă o mare asemănare și faptul că celula de scădere are în plus două porți NU care generează termenii a_i' și $(a_i \oplus b_i)'$.

Pornind de la această observație, se pune problema realizării unui circuit capabil să efectueze atât adunări cât și scăderi.

Construirea acestui circuit se bazează pe proprietatea de comutare a porții XOR, care constă în următoarea afirmație. Dacă se consideră o poartă XOR cu o intrare de date notată x și cu a doua intrare notată c atunci funcționarea ei este dată de următoarea tabelă de valori:

x	c	$y = x \oplus c$
0	0	0
0	1	1
1	0	1
1	1	0

și poate fi interpretată prin:

$$y = \begin{cases} x, & c = 0 \\ x', & c = 1 \end{cases}$$

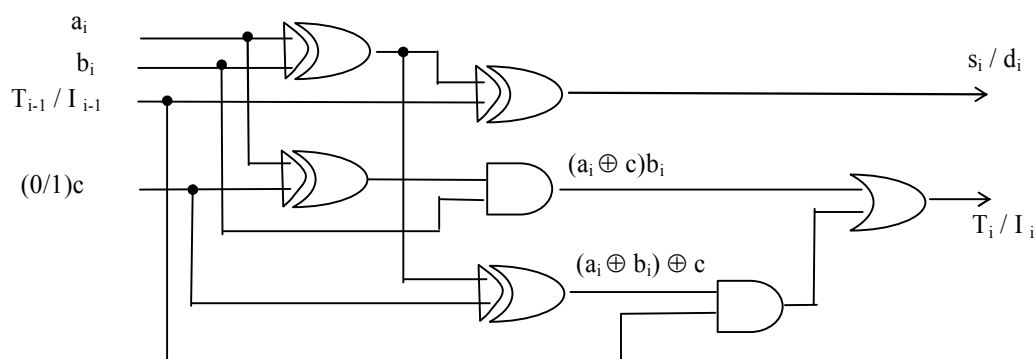
Astfel, putem denumi intrarea c ca fiind **intrare de comutare** (sau de condiție) deoarece valoarea ei este cea care decide dacă pe ieșirea din poarta XOR avem valoarea de pe intrare sau complementara (negarea) ei.

Cu acestea, se poate construi un circuit care să efectueze atât adunare cât și scădere, pe care îl vom numi **circuit sumator cu comutare**. Dacă pe intrarea c punem valoarea 0 atunci circuitul funcționează ca sumator, iar pentru $c = 1$, ca scăzător.

Semnificațiile pentru intrarea T_{i-1} / I_{i-1} și ieșirile s_i / d_i , T_i / I_i trebuie citite la *numărător*, respectiv la *numitor*. Astfel, dacă $c=0$ atunci valoarea pusă pe intrarea T_{i-1}/I_{i-1} reprezintă cifră de transport (T_{i-1}), iar pentru $c=1$ reprezintă cifră de împrumut (I_{i-1}). În consecință, valorile pe ieșiri se corespund astfel: s_i și T_i pentru $c=0$ și respectiv d_i și I_i pentru $c=1$.

Intrarea de comutare este folosită de două ori și anume pentru comutarea valorii lui a_i și a valorii $a_i \oplus b_i$. Prin comutarea valorii lui a_i se asigură că, pentru $c=0$, la ieșirea din poarta XOR avem a_i , iar pentru $c=1$, la ieșirea din poarta XOR avem a_i' . În acest fel, formula pentru valoarea depusă pe ieșirea a doua folosește pe a_i , dacă $c=0$ și pe a_i' , dacă $c=1$. Acest mod de calcul este corect și respectă formulele logice pentru T_i ($T_i = a_i b_i + \dots$), respectiv I_i ($I_i = a_i' b_i + \dots$).

Pentru operarea pe rangul i , acest circuit va consta din următoarele porți logice:



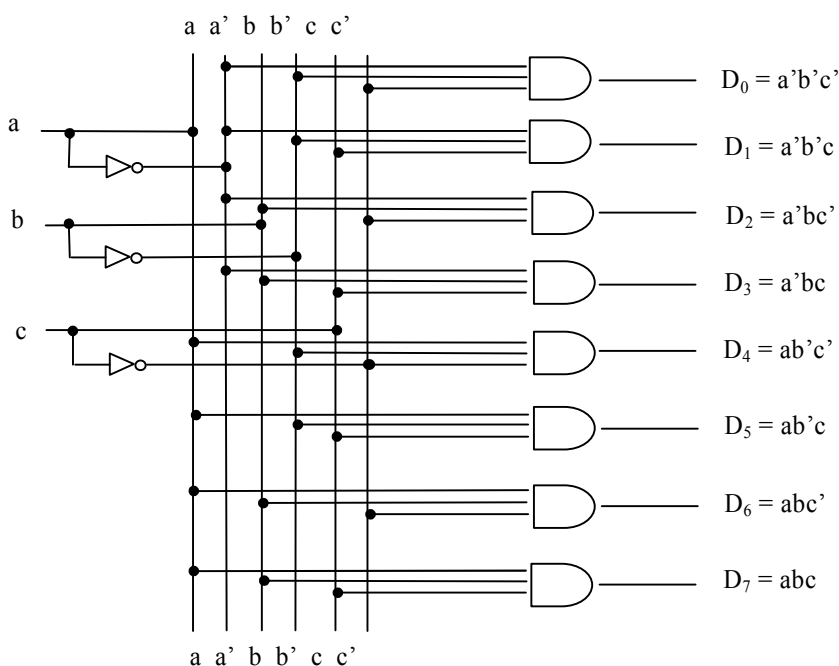
3.1.3 CLC decodare

În sistemul de calcul informația este reprezentată prin codificare binară. Un *cod binar de n biți* poate să reprezinte până la 2^n informații codificate distincte.

Un **decodor** este un circuit logic combinațional care convertește informația dată binar pe n intrări în cele 2^n ieșiri posibile. Dacă, dintre cele n intrări, unele sunt invalidate atunci decodorul va avea mai puțin de 2^n ieșiri.

Să considerăm un decodor cu trei intrări, a, b, c , toate valide. Rezultă că acesta va avea $8 = 2^3$ ieșiri. Algebric, funcțiile pentru cele 8 ieșiri sunt date de expresiile celor 8 termeni canonici de tip produs: $a'b'c'$, $a'b'c$, $a'bc'$, $a'bc$, $ab'c'$, $ab'c$, abc' , abc .

Un astfel de decodor se mai numește de tip 3×8 .

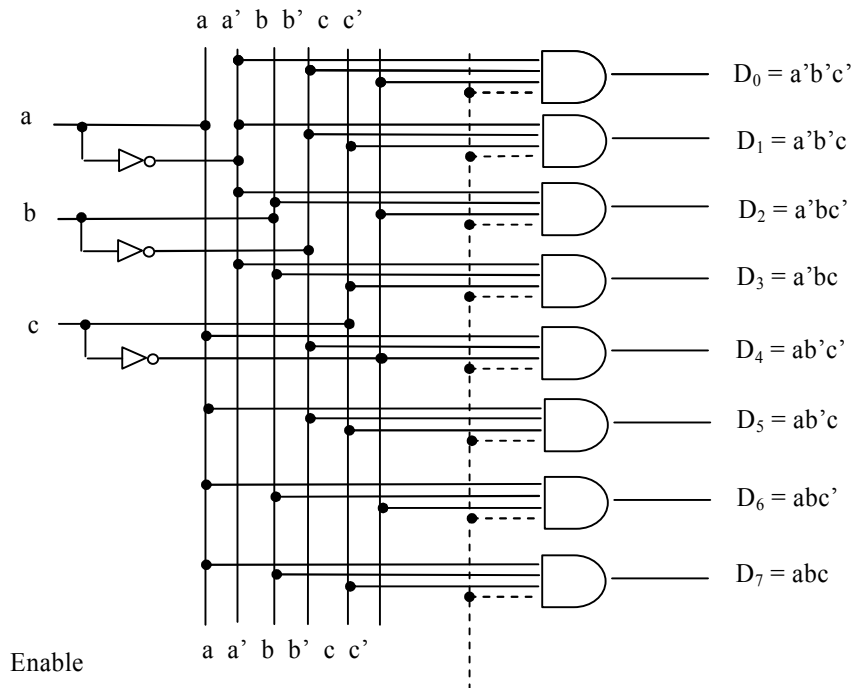


Deoarece termenii canonici implementați pe ieșiri sunt disjunctivi, rezultă că, indiferent de combinația valorilor pe intrări, o singură ieșire va avea valoare 1 logic, celelalte având valoarea 0 logic.

O aplicație imediată a acestui decodor este conversia numerelor din binar în octal. Concret, reprezentarea binară dată pe intrarea în decodor corespunde cifrei octale egală cu rangul ieșirii care are valoarea 1 logic.

abc	D ₀	D ₁	D ₂	D ₃	D ₄	D ₅	D ₆	D ₇
000	1	0	0	0	0	0	0	0
001	0	1	0	0	0	0	0	0
010	0	0	1	0	0	0	0	0
011	0	0	0	1	0	0	0	0
100	0	0	0	0	1	0	0	0
101	0	0	0	0	0	1	0	0
110	0	0	0	0	0	0	1	0
111	0	0	0	0	0	0	0	1

Pentru controlul funcționării unui decodor, de multe ori acesta este prevăzut cu o **intrare de validare** (*Enable*). Dacă pe această intrare avem 0 logic atunci circuitul nu funcționează adică, toate ieșirile sunt pe 0. Dacă pe intrarea *Enable* avem 1 logic atunci circuitul funcționează ca un decodor fără validare.



3.1.4 CLC multiplexoare

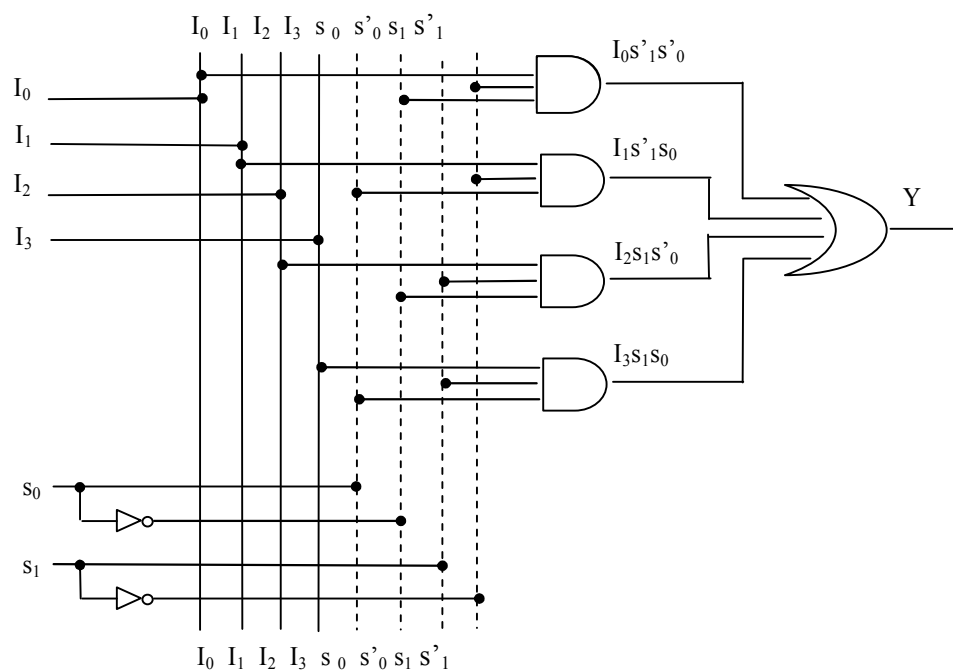
Un **multiplexor** este un circuit logic combinațional care primește informație binară de pe una din cele 2^n linii de date de intrare și o direcționează spre ieșirea sa unică. Linia de date este selectată prin n linii de selecție (linii de adresă).

Un multiplexor de tip $2^n \times 1$ are 2^n linii de date de intrare și n linii de selecție, ale căror combinații de biți determină care dintre intrările de pe liniile de date este selectată pentru ieșirea Y .

Selecția intrării se face conform cu următoarea tabelă (pe coloana a patra am trecut monoamele elementare care definesc selectarea ieșirii):

s_1	s_0	Y	
0	0	I_0	$s'_1 s'_0$
0	1	I_1	$s'_1 s_0$
1	0	I_2	$s_1 s'_0$
1	1	I_3	$s_1 s_0$

În figura următoare considerăm un multiplexor de tip 4×1 . Acest multiplexor are 6 intrări = 4 de date + 2 de selecție și o ieșire.



Observații. Datorită funcției de a selecta una dintre intrările în circuit, un multiplexor se mai numește și **selector de date** (*data selector*). Abrevierea pentru un multiplexor este **MUX**.

2.1.5 CLS bistabile

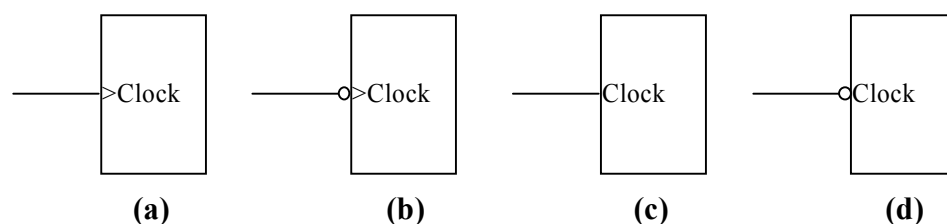
Cele mai multe dintre sistemele de calcul, pe lângă circuitele combinaționale (CLC) au nevoie (în special pentru componentele de memorare) și de circuite logice secvențiale (CLS). Dintre acestea, cele mai folosite sunt CLS sincrone sau cu sincronizare în timp. Un **CLS sincron** lucrează cu semnale care modifică conținutul zonelor de memorie numai la momente de timp discrete, bine determinate. Sincronizarea este asigurată de un generator de impulsuri de tact (de ceas) care pune ciclic pe intrarea de tact valorile 0 și 1.

Un **bistabil** este o structură de memorare bazată pe CLS sincrone. Un bistabil memorează un bit de informație.

Un bistabil își menține starea binară curentă până când primește un impuls de ceas care *intră* în ansamblul de porți logice și determină schimbarea de stare (eventual, schimbarea valorii memorate). Trecerea dintr-o stare în alta se face numai la primirea unui impuls de tact extern.

Dacă se ține cont de momentul în care se face efectiv schimbarea de stare a bistabilului (schimbarea valorii memorate), atunci în [Tanenbaum, pag. 134] se deosebesc două exemple de circuite bistabile: circuite bistabile (engl., *flip-flop*) și CBB adică circuite basculante bistabile (engl., *latch*). Pentru un bistabil schimbarea de stare are loc *în timpul* schimbării valorii logice pe intrarea de ceas, în timp ce pentru un CBB schimbarea de stare are loc *numai când* valoarea pe intrarea de ceas este deja schimbată. Cu alte cuvinte, un bistabil face schimbarea de stare pe durata transmiterii semnalului de tact (engl., *edge-trigger*), în timp ce un CBB face schimbarea de stare când semnalul de tact a ajuns la nivel (engl., *level-trigger*).

În figura următoare dăm reprezentarea standard propusă de Tanenbaum pentru: **(a)** bistabil cu tranziție pe front crescător (schimbarea de stare are loc când valoarea de pe intrarea de ceas trece din 0 în 1); **(b)** bistabil cu tranziție pe front descrescător (schimbarea de stare are loc când valoarea de pe intrarea de ceas trece din 1 în 0); **(c)** CBB cu încărcare pe 1 (schimbarea de stare are loc când valoarea de pe intrarea de ceas este 1); **(d)** CBB cu încărcare pe 0 (schimbarea de stare are loc când valoarea de pe intrarea de ceas este 0).



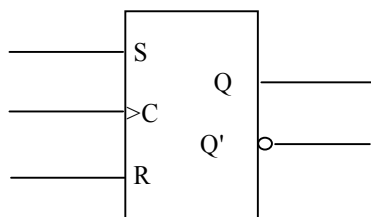
În cele ce urmează vom face referiri concrete la bistabilii cu tranziție pe frontul crescător.

Bistabilii se diferențiază prin numărul de intrări și prin felul în care valorile de intrare influențează starea bistabilului. Cele mai folosite tipuri de bistabili sunt: bistabil de tip SR; bistabil de tip D; bistabil de tip JK; bistabil de tip T.

Pentru început vom prezenta aceste circuite în cazul asincron, urmând ca, în final, să prezentăm și modelul sincron pentru bistabilul de bază, SR.

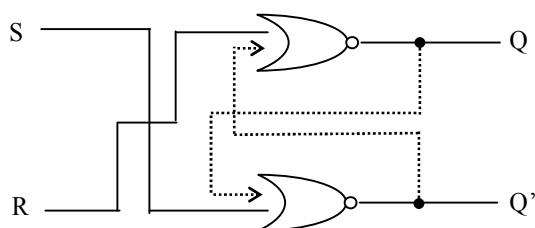
Bistabilul de tip SR are trei intrări notate respectiv cu: S pentru Setare, R pentru Resetare și C pentru intrarea de ceas și o ieșire, notată Q (quit). Auxiliar, un astfel de bistabil poate avea o ieșire suplimentară cu valoare egală cu ieșirea principală negată.

Simbolic, acest bistabil se reprezintă astfel:



Intrarea de ceas este marcată cu caracterul „>”. Un ceas este un circuit care emite o serie de pulsuri cu o durată de puls precisă și cu un interval precis între pulsuri succesive. Intervalul de timp între fronturile corespunzătoare a două pulsuri succesive se numește **durată de ciclu** (engl., *clock cycle time*). Frecvențele de puls (numărul de pulsuri în unitatea de timp) sunt în general între 1 și 500 MHz, corespunzând perioadei de ceas de la 1000 ns la 2 ns.

Porțile logice care asigură funcționarea acestui bistabil sunt două porți NOR interconectate. Prin *interconectare* înțelegem că ieșirile sunt repuse pe intrări prin *bucle de reacție*. În figura următoare buclele de reacție sunt figurate cu linie punctată.



Notăm cu Q_t valoarea pe ieșirea Q a bistabilului la momentul t și cu Q_{t+1} valoarea pe aceeași ieșire la momentul următor, $t+1$. Cu alte cuvinte, Q_t dă starea bistabilului înainte de aplicarea semnalelor S și R pe intrare, iar Q_{t+1} dă starea bistabilului după aplicarea semnalelor respective.

Datorită existenței buclelor de reacție, rezultă că starea Q_{t+1} depinde de intrările S și R, dar și de starea anterioară, Q_t .

Conform legării porților logice în reprezentarea de mai sus, expresia logică pentru Q_{t+1} este

$$Q_{t+1} = (R + (S + Q_t))' = R' (S + Q_t) = R'S + R'Q_t.$$

Această expresie determină următoarea tabelă de valori pentru schimbarea de stare ($Q_t \rightarrow Q_{t+1}$) a bistabilului de tip SR:

S	R	Q_t	Q_{t+1}	
0	0	0	0	fără modificare de stare
0	0	1	1	(Q_{t+1} coincide cu Q_t)
0	1	0	0	punere pe 0 (resetare)
0	1	1	0	
1	0	0	1	punere pe 1 (setare)
1	0	1	1	
1	1	0	0(*)	stare nedeterminată
1	1	1	0(*)	(comenzi interzise)

Observație. Pentru intrarea (1,1), funcționarea teoretică a circuitului determină valoarea 0 pentru Q_{t+1} , indiferent de starea anterioară a bistabilului, Q_t . Cu toate acestea, practic, datorită întârzierilor care apar în sistem la transmiterea semnalelor, starea Q_{t+1} este nedeterminată (nu se stabilizează, chiar după mai multe reveniri pe buclele de reacție). În acest caz spunem că *funcționarea bistabilului este ambiguă*.

Din tabelul anterior rezultă cum depinde starea următoare Q_{t+1} de valorile puse pe intrările în bistabil, S și R.

S	R	Q_{t+1}	
0	0	Q_t	fără modificare de stare
0	1	0	resetare = punere pe 0
1	0	1	setare = punere pe 1
1	1	?	situație imprecisă (nedeterminată)

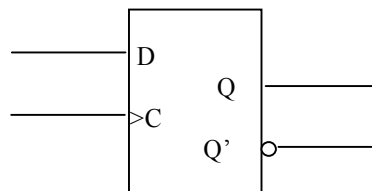
Din afirmațiile anterioare se poate trage concluzia că un bistabil de tip SR nu funcționează întotdeauna corect. De aceea, se impune stabilirea unei relații pe care trebuie să o verifice valorile puse pe intrările S și R astfel încât bistabilul să nu intre într-o stare nedeterminată. Această ecuație este

$$SR = 0$$

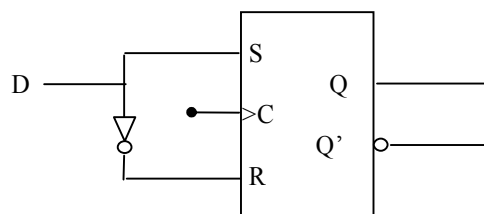
și ea asigură că cel puțin una dintre valorile de intrare este 0. Cu alte cuvinte, se evită tocmai situația în care ambele intrări sunt pe 1 logic.

Bistabilul de tip D (Data) este ușor modificat față de bistabilul – SR. Acesta are o singură intrare de date (D), intrarea de tact (C) și ieșirile identice cu tipul SR. Pe intrarea D se inserează un inversor, astfel că se obțin două direcții care vor corespunde intrărilor S și R ale tipului SR.

Simbolul este:



Un bistabil de tip D se obține din SR prin următoarea modificare:



Din această modalitate de a lega intrarea D la intrările S și R rezultă:

- ✓ a avea intrarea D în 0 logic este echivalent cu a avea 0 pe intrarea S și 1 pe intrarea R;
- ✓ a avea intrarea D în 1 logic este echivalent cu a avea 1 pe intrarea S și 0 pe intrarea R.

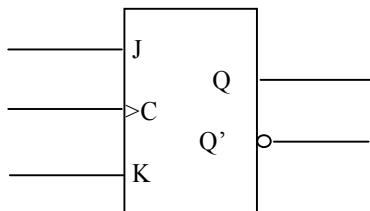
Se observă că oricare ar fi valoarea pusă pe intrarea în bistabilul de tip D, acesta funcționează corect, deoarece este întotdeauna verificată relația

$$SR = 0.$$

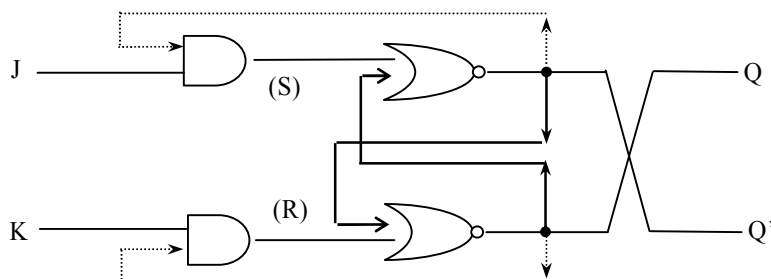
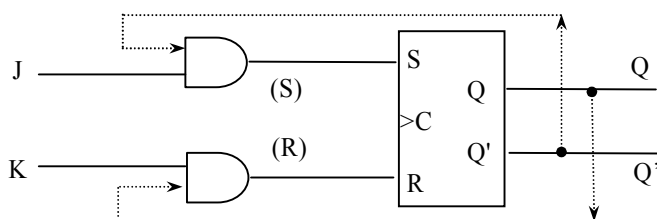
Funcționarea bistabilului de tip SR determină următoarea tabelă de funcționare pentru bistabilul de tip D:

D	Q_{t+1}
0	0 resetare = punere pe 0
1	1 setare = punere pe 1

Bistabilul de tip JK are două intrări de date, J și K, intrarea de tact, C și ieșirile identice cu tipul SR.



Din punctul de vedere al circuitelor interne, un bistabil JK este unul SR completat cu două porți AND și două *linii de reacție totală*. În figurile următoare liniile de reacție totală sunt marcate cu linie punctată. Prezentăm structura bistabilului de tip JK lăsa două nivele de detaliere:



Această structură internă determină schimbarea de stare $Q_t \rightarrow Q_{t+1}$ conform cu următorul tabel:

J	K	Q_t	Q_{t+1}	
0	0	0	0	fără modificare de stare (Q_{t+1} coincide cu Q_t)
0	0	1	1	
0	1	0	0	punere pe 0 (resetare)
0	1	1	0	
1	0	0	1	punere pe 1 (setare)
1	0	1	1	
1	1	0	1	trecere în starea complementară
1	1	1	0	

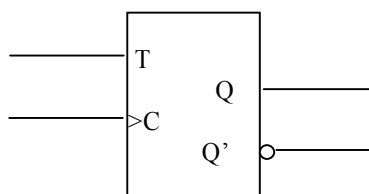
Corespunzător, tabela de funcționare pentru bistabilul de tip JK este următoarea:

J	K	Q_{t+1}	
0	0	Q_t	Fără modificare de stare
0	1	0	Resetare = punere pe 0
1	0	1	Setare = punere pe 1
1	1	Q'_t	Trecere în starea complementară

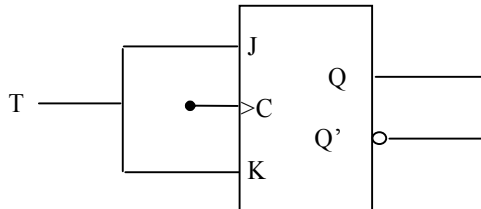
Observație. Despre bistabilul de tip JK putem spune că este primul model de circuit secvențial veritabil. Aceasta deoarece, când avem (1,1) pe intrările J și K, starea următoare Q_{t+1} se poate determina numai dacă se cunoaște starea anterioară Q_t .

Bistabilul de tip T (Toggle = piron, cârjă) este un model de bistabil care funcționează în doi timpi. Această afirmație va fi pe deplin înțeleasă după prezentarea funcționării circuitului.

Simbolul de reprezentare este:



Un bistabil de tip T se obține din unul JK prin legarea intrărilor J și K la aceeași intrare T, astfel:

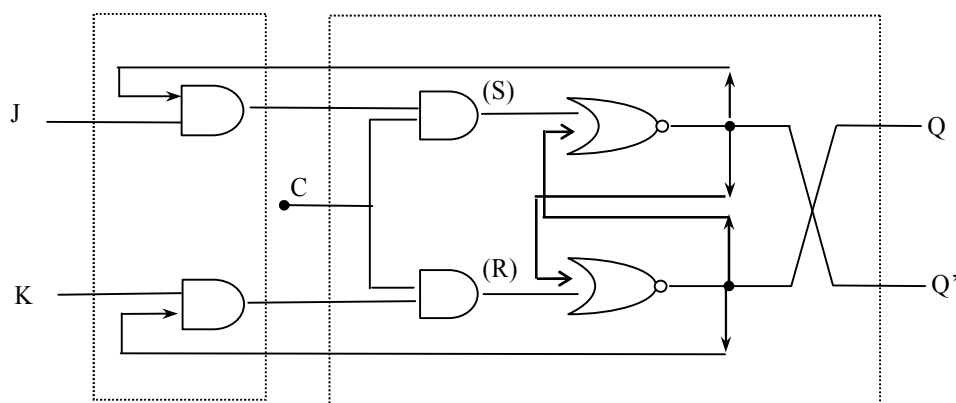


Din această construcție rezultă că bistabilul de tip T are două variante de funcționare, care coincid cu tipul JK pentru $J = K = 0$ și $J = K = 1$:

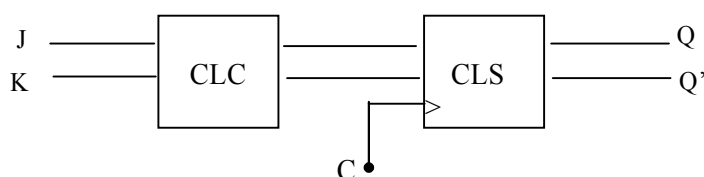
T	Q_{t+1}	
0	Q_t	Fără modificare de stare
1	Q'_t	Trecere în starea complementară

Observație. Cât timp pe intrarea T avem 1 logic, circuitul intră în starea complementară a stării curente și, apoi, la următoarea schimbare de stare, revine în starea inițială. De aceea, se spune despre acest bistabil că funcționează în doi timpi.

În încheierea paragrafului referitor la bistabili prezentăm modelul sincron pentru bistabilul de tip JK. Se va vedea că acesta completează modelul asincron cu două porți SI care preiau valoarea logică pusă pe intrarea de tact, C.



În această reprezentare am pus în evidență și secțiunile asincronă și sincronă ale circuitului, care corespund celor două componente de bază: un CLC și un CLS.



3.1.6 Circuite integrate

În anii '50 se descoperă proprietățile semiconductoare ale siliciului și apar astfel primele dispozitive semiconductoare, numite tranzistoare. Generația a treia de calculatoare (anii 1964-1981) folosește circuite integrate obținute prin imprimarea tranzistorilor pe cipuri de siliciu. Ulterior, perfecționarea circuitelor integrate a constat în creșterea densității de componente incluse pe un cip.

Componentele digitale ale unui sistem de calcul sunt realizate cu circuite logice integrate. Un **circuit integrat** (CI) este format dintr-un semiconductor de siliciu, de dimensiuni foarte reduse, numit **chip**, care conține componentele porților logice. Mai multe porți logice sunt interconectate în interiorul chip-ului astfel încât să formeze circuitul cerut. Chip-ul este închis într-o carcasă de ceramică sau de material plastic, iar conexiunile sunt sudate cu conductoare fine de aur la pinii de ieșire pentru a forma circuitul integrat. Numărul pinilor poate varia de la 14 pini la un pachet integrat mic, până la 1000 sau mai mulți la un circuit mare.

Conexiunile pastilei de siliciu pot fi la alte circuite, la sursele de tensiune, la masa electrică. În funcție de tensiunea care codifică valorile logice, circuitul integrat lucrează în *logică pozitivă* sau în *logică negativă*. Un circuit lucrează în *logică pozitivă* dacă valoarea 1 logic corespunde unei tensiuni pozitive (3V, 12V, 15V), iar 0 logic corespunde unui nivel coborât de tensiune (0V, -12V, respectiv -15V). Un circuit lucrează în *logică negativă* dacă valoarea 1 logic corespunde unui nivel scăzut de tensiune (0V, -12V, -15V), iar 0 logic corespunde unui nivel ridicat de tensiune (3V, 12V, 15V).

O dată cu îmbunătățirea tehnologiilor de realizare a circuitelor integrate, a crescut considerabil și numărul porților care pot fi implementate pe un același chip (densitatea de integrare). După acest criteriu, diferența între chip-urile care au câteva porți și cele care au sute de porți componente se face prin numele acestora. Astfel, avem următoarele clase: CI pe scară joasă, CI pe scară medie, CI pe scară largă și CI pe scară foarte largă. Scara de integrare se referă la numărul de componente electronice pe unitatea de suprafață a circuitului integrat.

Dăm în tabelul următor câteva caracteristici pentru aceste clase:

SSI	<i>Small Scale Integration</i>	mai puțin de 10 porți	IN-OUT sunt conectate direct la pinii circuitului
MSI	<i>Medium Scale Integration</i>	10-200 porți	folosite în decodare, sumatoare, regiștri
LSI	<i>Large Scale Integration</i>	200-câteva sute de porți	includ procesoare, chipuri de memorie, module programabile
VLSI	<i>Very Large Scale Integration</i>	mii de porți	dimensiuni mici, preț redus, deci foarte folosite în sistemele moderne.

Circuitele logice integrate se pot clasifica și după tehnologia de fabricare, rezultând *familii de circuite logice*. Fiecare familie logică de circuite este reprezentată de un tip de circuit pe baza căruia se dezvoltă toate circuitele din acea familie. Comercial vorbind, structurile care s-au impus sunt: CI de tip TTL (*Transistor – transistor logic*), CI de tip ECL (*Emitter – coupled logic*) și CI de tip MOS (*Metal – oxide semiconductor*).

Tehnologia principală de realizare a circuitelor integrate moderne este *TTL – logică pozitivă* cu 0V pentru 0 logic și 3V pentru 1 logic.

Caracterizarea acestor tipuri de circuite integrate poate fi făcută astfel:

TTL	originea = DTL (diode – transistor logic) care implementa poarta NAND cu diode și tranzistori; ulterior s-au înlocuit diodele cu tranzistori și, de aceea repetăm cuvântul tranzistor în denumire	tensiunea maximă aplicabilă este de 5V, iar valorile logice 0 și 1 corespund pentru 0V și respectiv 3.5V;
ECL	folosite în sisteme de viteză foarte mare (supercalculatoare, procesoare de semnale)	tranzistorii operează într-o stare nesaturată, condiție care determină o întârziere în propagarea semnalului de maxim 1-2 nanosecunde;
MOS	este un tip de CI care folosește tranzistori unipolari, față de TTL și ECL care folosesc tranzistori bipolarari.	

În funcție de numărul valorilor logice recunoscute la terminalele unui circuit integrat avem:

- ✓ circuitele integrate cu două valori logice – aceste valori sunt 0 și 1; astfel de circuite sunt circuitele TTL standard;
- ✓ circuite integrate cu trei valori logice – aceste valori sunt 0, 1 și Z (starea de înaltă impedanță; astfel de circuite sunt circuitele TSL (engl., *Three State Logic*) care permit conectarea de tip magistrală (ieșirile și intrările mai multor porți logice pot fi conectate la aceleași linii de semnal).

Rezumat.

Acest paragraf are ca obiectiv principal definirea principalelor circuite logice care intră în structura unui circuit elementar de memorie (bistabil), a unui registru (vezi paragraful al doilea) și/sau în structura unei UAL elementare. Tipurile de circuite sunt prezentate gradat, de la simplu la complex, așa cum se poate vedea și din lista de cuvinte cheie.

Pentru toate aceste prezentări teoretice de modele, în culegerea de probleme sunt date exemple concrete de utilizare și interconectare a circuitelor. Structura culegerii de probleme urmărește paragraf cu paragraf cursul, astfel încât studenții pot rezolva problemele pas cu pas, pe măsură ce sunt parcurse la curs elementele teoretice. Ultimul paragraf al culegerii de probleme conține o serie de exemple de sinteză, unele fiind chiar modele concrete de circuite interne ale calculatorului.

Cuvinte cheie.

circuit logic
poartă logică, poartă logică elementară, poartă logică uzuală
CLC, CLC aritmetic (sumator, scăzător, sumator cu comutare),
 decoder, multiplexor
CLS, CLS bistabil (SR, D, JK, T)
circuit integrat

NOTIȚE

2.2. Registre

Un **registru** păstrează temporar o informație codificată binar.

Un **registru** este un ansamblu de bistabili (circuite basculante bistabile, CBB) și, eventual, porți logice care realizează schimbările de stare ale bistabililor. Grupul de bistabili care formează registrul pot fi încărcăți simultan, sub acțiunea unui impuls de tact unic.

Un **registru de n biți** are n bistabili și poate memora orice informație reprezentabilă pe n biți. În acest caz spunem că *registrul este de capacitate n* , respectiv că *n este lungimea cuvântului registru*. Porțile logice au rolul de a controla când și cum o informație nouă este transferată în registru.

Componentele unui sistem digital sunt complet definite de regiștrii pe care îi conțin și de operațiile care se execută asupra datelor lor. În calculatoarele moderne registrele sunt folosite ca locații de memorie rapidă, păstrând informații care urmează să fie prelucrate. Informațiile memorate pot fi: date, adrese, coduri de instrucțiune, operanzi, rezultate ale unor operații, informații de stare etc. Prelucrarea datelor la nivel de registru poate consta în efectuarea unor operații sau transferul de date între registre.

Operațiile executate asupra datelor memorate în regiștri se numesc **microoperații**. Exemple: încărcare (serială sau paralelă), deplasare (la stânga sau la dreapta), rotație (spre stânga sau spre dreapta), numărare, ștergere.

Pentru executarea fiecăreia dintre aceste microoperații există structuri bine determinate ale regiștrilor respectivi. În paragrafele următoare vom descrie structurile unora dintre acești regiștri.

3.2.1 Registre de microoperație

3.2.1.1 Registru cu încărcare paralelă

Funcția unui registru cu încărcare paralelă este de a prelua valorile logice puse pe intrările de date, simultan pentru toate pozițiile binare și la primirea aceluiași impuls de ceas.

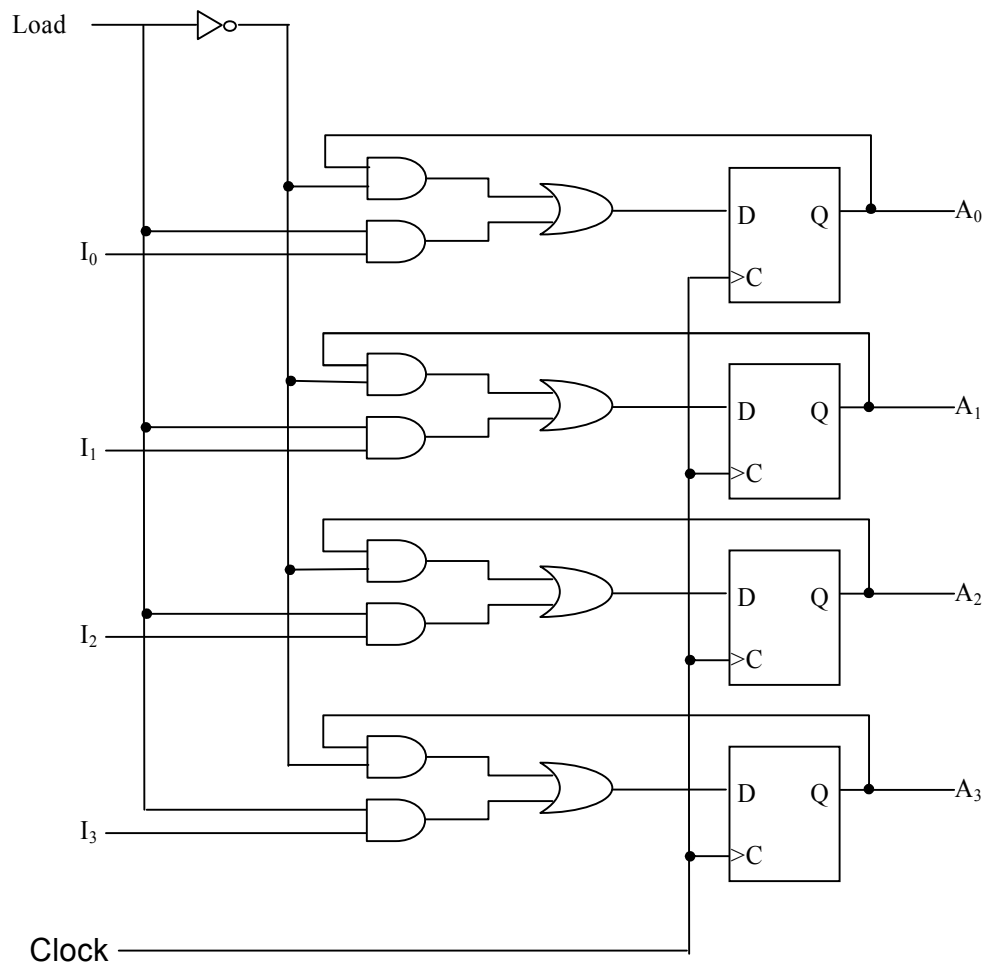
Acest registru constă din patru bistabili de tip D, legați prin porți logice de cele patru intrări de date I_0, I_1, I_2, I_3 și de intrarea de validare, Load.

Rolul intrării Load este de a dirija preluarea datelor de pe intrările I_0, I_1, I_2, I_3 numai la primirea unui impuls de tact. Dacă pe intrarea Load avem 0 logic atunci intrările de date sunt inhibitate și intrările D în bistabili sunt conectate respectiv cu ieșirile Q. Dacă pe Load avem 1 logic atunci valorile depuse pe intrările I_0, I_1, I_2, I_3 , sunt încărcate în registru.

Valorile logice considerate a fi memorate în registru în fiecare moment sunt cele de pe liniile A_0, A_1, A_2, A_3 .

Structura unui astfel de registru este dată în figura următoare.

Bistabilul de tip D nu are variantă de funcționare *fără modificare de stare* ci, la fiecare impuls de tact primit, valoarea preluată de bistabil pe intrarea D determină următoarea stare la ieșire. Pentru a păstra valoarea la ieșire nemodificată, cât timp nu se primește impuls de tact, este necesar să menținem intrarea D egală cu valoarea curentă de pe ieșire. Această funcționare este asigurată de *conexiunile de reacție*, care leagă fiecare ieșire Q la intrarea corespunzătoare în poartă logică ȘI și, mai departe la intrarea D a aceluiași bistabil.

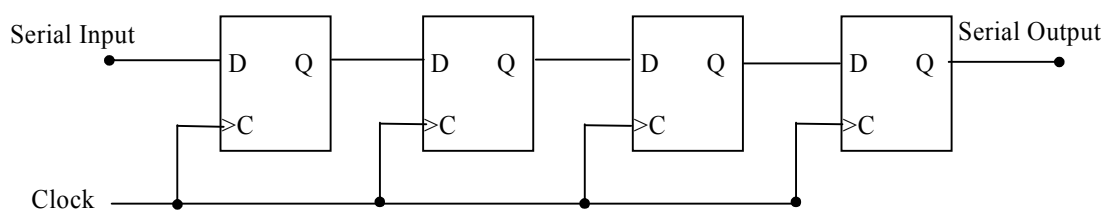


Impulsul de tact se aplică pe intrarea Clock conform cu frecvența generatorului de tact. Cu toate acestea, dacă circuitul este dotat cu o intrare de validare de tip Load, atunci ea este cea care decide dacă următorul impuls de ceas va determina încărcarea simultană a datelor depuse pe intrările de date sau preluarea pe liniile de reacție a valorilor curente memorate în registru. Cu alte cuvinte, intrarea Load decide dacă registrul primește informație nouă sau informația memorată rămâne nemodificată.

3.2.1.2 Registru de deplasare

Un registru capabil să deplaseze, adică să translateze cu o poziție informația memorată de bistabilii săi se numește **registru de deplasare**. În funcție de construcția registrului, operația de mutare (*shift*) se poate face spre stânga, spre dreapta sau în ambele direcții.

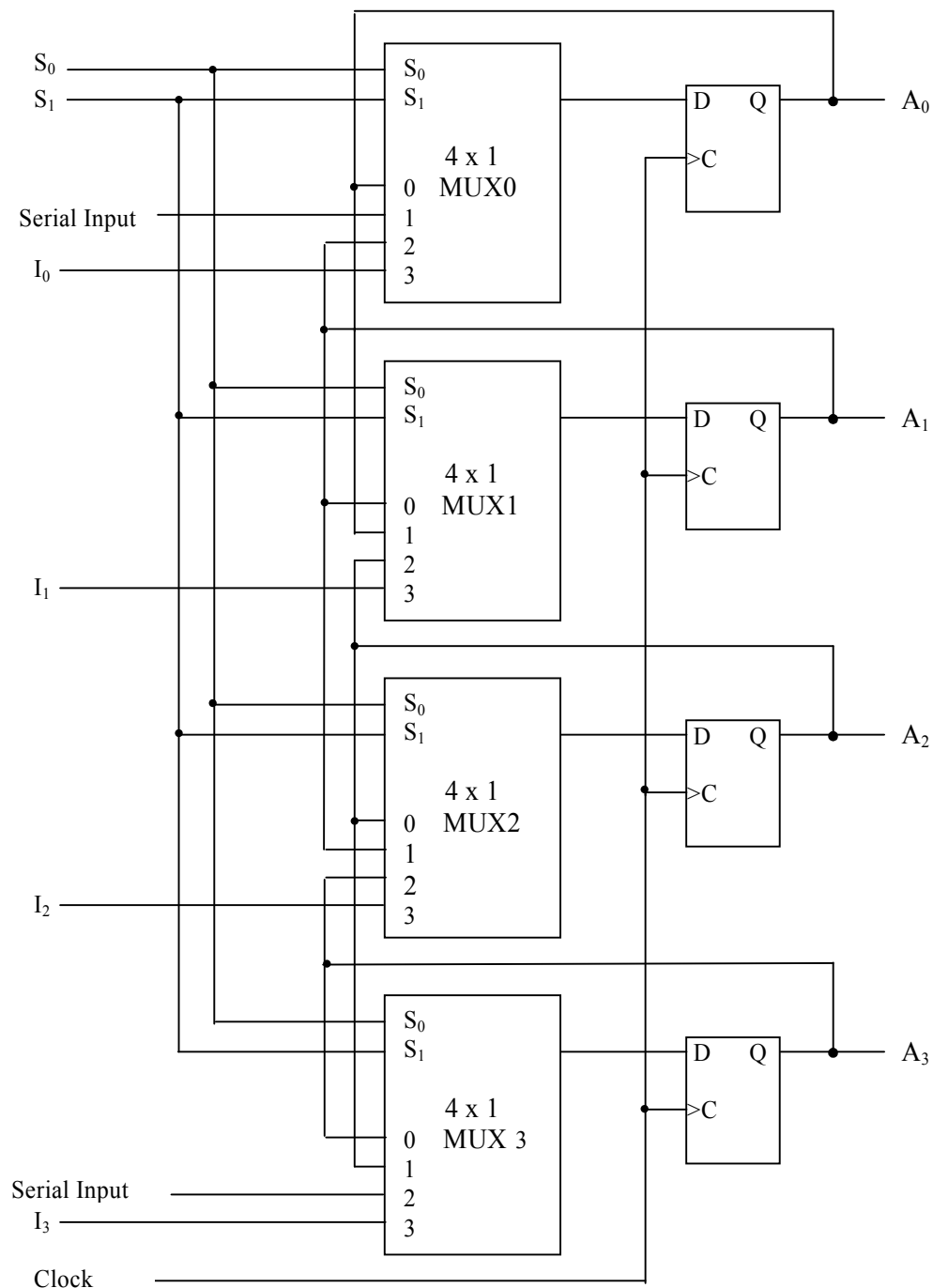
Reprezentarea este:



Se observă că ieșirea dintr-un bistabil este intrare în următorul. În plus, funcționarea este sincronizată prin faptul că toate intrările de ceas sunt legate la aceeași sursă. Astfel, toți bistabilii vor primi la același moment impulsul de tact care determină deplasarea informației.

Intrarea *Serial Input* conține informația care se va memora în primul bistabil, iar ieșirea *Serial Output* va conține vechiul conținut al ultimului bistabil al registrului de deplasare analizat.

Detaliem acest tip de registru prezentând în figura următoare structura internă a unui registru pe 4 biți care poate executa sau o deplasare, sau o încărcare paralelă (simultană), în funcție de valorile puse pe intrările de selecție ale circuitului. Registrul constă din patru multiplexoare conectate la patru bistabili de tip D.



Reamintim că valorile memorate în registru la un moment dat sunt cele de pe liniile A_0 , A_1 , A_2 , A_3 și ele se modifică numai simultan, la primirea unui nou impuls de tact. Cele două intrări *Serial Input* introduc valori noi în registru pe pozițiile eliberate de operația de deplasare.

Din schemă se observă că:

- fiecare MUX preia pe intrarea sa 0 valoarea curentă din registru, transmisă pe linia de reacție;

- fiecare MUX preia pe intrarea sa 1 valoarea din locația de rang cu 1 mai mic;
- fiecare MUX preia pe intrarea sa 2 valoarea din locația de rang cu 1 mai mare;
- fiecare MUX preia pe intrarea sa 3 valoarea depusă pe intrarea de date corespunzătoare.

Această organizare este în concordanță cu următoarea tabelă de funcționare. În funcție de valorile depuse pe intrările de selecție S_0 și S_1 , avem:

S_1	S_0	Intrarea selectată de fiecare MUX	
0	0	0	Fără modificare de stare
0	1	1	Shift Right (down)
1	0	2	Shift Left (up)
1	1	3	Încărcare paralelă

3.2.1.3 Numărătoare binare

O numărătoare este un registru care, la aplicarea impulsurilor de ceas, urmează o succesiune predefinită de stări. Numărul de stări distincte definește *modulul de numărare*.

În particular, numărătoarele binare vor lucra pe secvențe de valori binare. Aplicațiile imediate ale unui registru numărătoare sunt:

- contorizarea numărului de apariții ale unui eveniment;
- cronometrarea necesară generării semnalelor de control pentru secvențele de operații de executat în sistem.

În funcție de sensul de numărare, avem: numărătoare directă, numărătoare inversă, numărătoare reversibilă, numărătoare ciclică.

Pentru o numărătoare binară pe 4 biți succesiunea de stări prin care trece este următoarea: 0000 → 0001 → 0010 → 0011 → 0100 → 0101 → 0110 → 0111 → 1000 → 1001 → 1010 → 1011 → 1100 → 1101 → 1110 → 1111.

Realizarea circuitului de numărare constă în transcrierea cu porți logice a următoarei observații. Pentru a trece dintr-o stare în alta trebuie efectuate operațiile:

1. LSB (bitul cel mai puțin semnificativ) se complementează;
2. fiecare dintre ceilalți biți se complementează numai dacă în configurația anterioară a avut toți biții de rang mai mic egali cu 1.

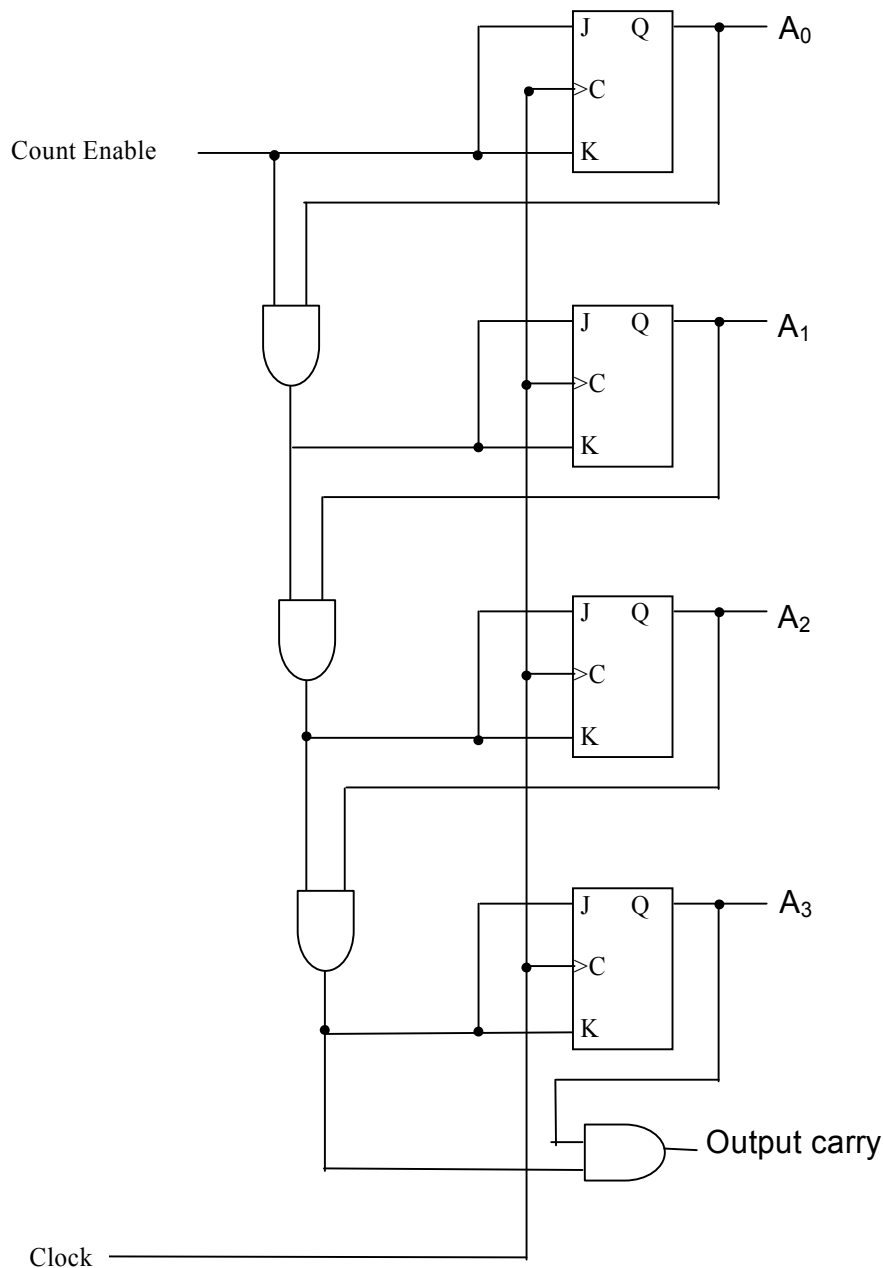
De aici rezultă că acest registru trebuie să folosească bistabili care au posibilitatea de a trece în starea complementară a stării curente, deci bistabili de tip JK sau T.

În figura următoare este reprezentat un registru de numărare pe patru biți.

Intrarea *Count Enable* este una de validare a funcționării registrului, deci decide care sunt impulsurile de tact care determină trecerea numărătoarei din starea curentă în starea următoare. Așa cum este construit circuitul, schimbarea de stare are loc când pe *Count Enable* avem 1 logic.

Conform cu operația a doua care trebuie efectuată la orice schimbare de stare, valoarea curentă din fiecare locație a registrului influențează valoarea încărcată în fiecare dintre pozițiile de rang mai mare. Această proprietate este asigurată prin faptul că fiecare linie A_i este legată „în cascadă” prin porți SI la intrările în toți bistabilii de rang mai mare.

Ieșirea *Output Carry* are semnificație de cifră de transport. Aceasta este folosită în situația în care configurația curentă este 1111 și numărătoarea ar trebui să treacă în configurația următoare, 10000. Având numai patru biți, registrul va reține 0000, iar bitul 1 va fi transmis pe ieșirea de transport, *Output Carry*.



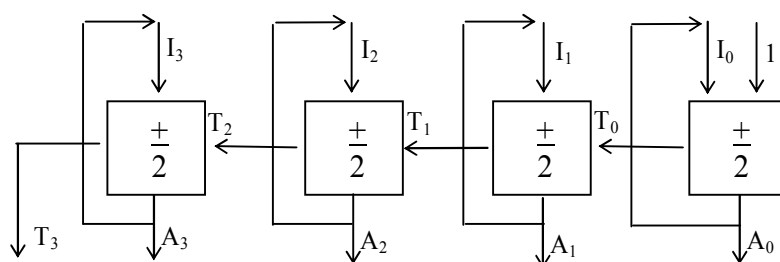
3.2.1.4 Registru de incrementare

Un registru de incrementare reprezintă fizic microoperația de incrementare. Prin aceasta, valoarea următoare care se încarcă în registru este egală cu valoarea anterioară plus 1.

Pentru realizarea unui registru de incrementare se poate folosi structura unei numărătoare binare sau se poate construi un CLC de incrementare.

Dacă se efectuează o microoperație de incrementare pe n biți atunci circuitul logic combinațional de incrementare este un sumator format din n semisumatoare, dintre care cel corespunzător rangului zero are o intrare egală cu 1.

Pentru CLC de incrementare pe 4 biți reprezentarea este:



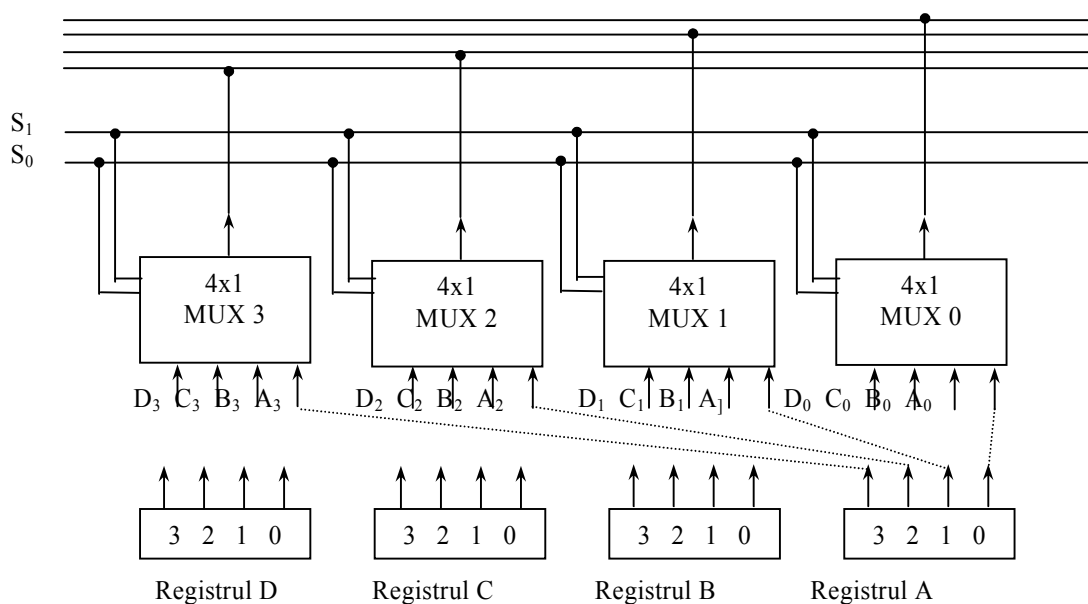
3.2.2 Transferul datelor din regiștri pe magistrală

Un calculator digital trebuie să fie prevăzut cu căi de transfer pentru informațiile memorate în regiștri. Pentru aceasta, o schemă eficientă este sistemul cu **magistrală comună**.

O structură de magistrală constă într-un set de linii comune, câte una pentru fiecare bit al unui registru, prin care informația binară este transferată la momente de timp predefinite. Pentru fiecare transfer, se selectează un registru prin semnale de control depuse pe magistrală.

O modalitate de a construi un sistem cu magistrală comună folosește multiplexoare. Aceștia selectează informația din registrul sursă, care este apoi depusă pe magistrală.

Reprezentarea acestui sistem este dată în figura următoare:



Se observă că MUX0 multiplexează cei patru biți 0 ai regiștrilor, MUX1, biții 1, MUX2 biții 2 și MUX3, biții 3. Ieșirile celor patru multiplexoare formează **magistrala comună cu patru linii de date**.

În funcție de combinațiile pe intrările de selecție, pe magistrală se depune conținutul unui registru sau al altuia astfel:

S1	S0	Register selected
0	0	A
0	1	B
1	0	C
1	1	D

Rezumat.

Considerând cunoscute structurile de circuite introduse anterior, în acest paragraf sunt descrise structurile interne la nivel de poartă logică pentru principalii regiștri de microoperație.

Pentru fiecare astfel de registru se va insista în mod special pe înțelegerea modului de funcționare ca circuit secvențial (Funcționarea unui circuit secvențial este corect definită ca o secvență de stări). De asemenea, un pas important în înțelegerea acestor modele constă în identificarea rolului fiecărui circuit elementar în ansamblul constructiv al unui registru și, în plus, studentul trebuie să poată răspunde concret, în fiecare caz în parte, la întrebări de genul: care este rolul în circuit a fiecărei componente, se poate înlocui o componentă internă cu o alta, care este câștigul/pierderea în cazul unei astfel de înlocuiri ș.a.m.d.

Cuvinte cheie.

registru
microoperație (încărcare, deplasare, numărare, incrementare)
transfer de date

NOTIȚE



PROBLEME REZOLVATE

1. Introducere



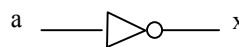
În introducerea acestei mini-culegeri de probleme pentru disciplina ARHITECTURA CALCULATOARELOR vă amintim câteva dintre elementele de bază prezentate la curs.

În cele ce urmează, funcțiile logice elementare sunt reprezentate astfel:





- funcția SI – multiplicativ;
- funcția SAU – aditiv;
- funcția NU – prin operatorul apostrof atașat variabilei care se neagă.

Pentru funcția logică XOR (SAU EXCLUSIV) vom folosi simbolul \oplus .

Porțile logice elementare, adică simbolurile grafice folosite pentru reprezentarea circuitelor logice asociate funcțiilor logice elementare sunt:

<i>Nume</i>	<i>Simbol grafic</i>	<i>Expresie logică</i>
ȘI LOGIC		$x = a \cdot b$ sau $x = a b$
SAU LOGIC		$x = a + b$
NU LOGIC		$x = a'$

Porțile logice asociate funcțiilor logice uzuale sunt:

<i>Nume</i>	<i>Simbol grafic</i>	<i>Expresie logică</i>
NAND nu – și		$x = (a b)'$
NOR nu – sau		$x = (a + b)'$
XOR sau exclusiv (anticoincidență)		$x = a'b + ab' =$ $= a \oplus b$
NXOR NOR exclusiv (echivalență) (coincidență)		$x = ab + a'b' =$ $= (a \oplus b)'$

Simplificarea funcțiilor logice se bazează pe un minim de identități din algebra booleană:

$$(1) \quad a + 0 = a$$

$$(9) \quad a \cdot 0 = 0$$

$$(2) \quad a + 1 = 1$$

$$(10) \quad a \cdot 1 = a$$

$$(3) \quad a + a = a$$

$$(11) \quad a \cdot a = a$$

$$(4) \quad a + a' = 1$$

$$(12) \quad a \cdot a' = 0$$

$$(5) \quad a + b = b + a$$

$$(13) \quad ab = ba$$

$$(6) \quad a + (b + c) = (a + b) + c$$

$$(14) \quad a(bc) = (ab)c$$

$$(7) \quad a(b + c) = ab + ac$$

$$(15) \quad a + bc = (a + b)(a + c)$$

$$(8) \quad (a + b)' = a'b'$$

$$(16) \quad (ab)' = a' + b'$$

$$(17) \quad (a')' = a$$

NOTIȚE

2. Probleme cu circuite logice combinaționale aritmetice

2.1. Problema 1

Se consideră **funcția majoritate pe trei variabile**. Dacă cele trei variabile de intrare sunt notate a, b, c atunci expresia de definiție pentru funcția M , majoritate este următoarea:

$$M(a,b,c) = \begin{cases} 1, & \text{dacă majoritatea valorilor de intrare sunt 1} \\ 0, & \text{în rest.} \end{cases}$$

Se cere:

- să se completeze tabela de valori a funcției M ;
- să se deducă expresia logică a funcției M ;
- să se construiască circuitul logic asociat funcției M .

Rezolvare

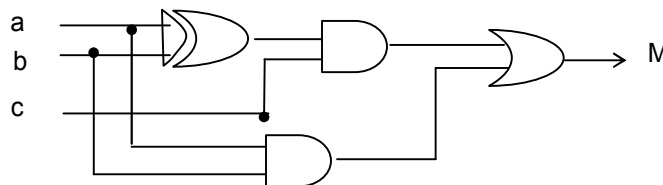
Tabela de valori a funcției majoritate M este:

A	b	c	M
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Expresia logică a funcției M este:

$$M = a'bc + ab'c + \underline{abc'} + \underline{abc} = (a'b + ab')c + ab(c' + c) = (a \oplus b)c + ab.$$

Circuitul logic pentru funcția M poate fi reprezentat astfel:



Observații

Expresia funcției M este unică abstracție făcând de o permutare a variabilelor. Această afirmație este justificată de faptul că, în funcție de gruparea termenilor pentru simplificare, expresia lui M poate fi:

$$M = (a \oplus b)c + ab = (b \oplus c)a + bc = (c \oplus a)b + ca.$$

Expresia funcției majoritate M este similară cu expresia cifrei de transport T_i de la rangul i la rangul $i+1$ pentru circuitul sumator complet.

2.2. Problema 2

Se consideră **funcția de paritate pe trei variabile**. Dacă cele trei variabile de intrare sunt notate a, b, c atunci expresia de definiție pentru funcția P , de paritate este următoarea:

$$P(a,b,c) = \begin{cases} 1, & \text{dacă numărul valorilor de intrare egale cu 0 este par} \\ 0, & \text{în rest.} \end{cases}$$

Se cere:

- să se completeze tabela de valori a funcției P;
- să se deducă expresia logică a funcției P;
- să se construiască circuitul logic asociat funcției P.

Rezolvare

Tabela de valori a funcției de paritate P este: (considerăm că 0 este număr par)

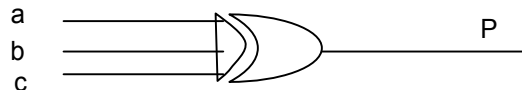
a	b	c	P
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

Expresia logică a funcției P este:

$$P = \underline{a'b'c} + a'bc' + ab'c' + \underline{abc} = (a'b' + ab)c + (a'b + ab')c' =$$

$$= (a \oplus b)'c + (a \oplus b)c' = c \oplus (a \oplus b).$$

Circuitul logic pentru funcția P poate fi reprezentat astfel:



Observații

Expresia funcției P este unică abstracție făcând de o permutare a variabilelor. Această afirmație este justificată de faptul că, în funcție de gruparea termenilor pentru simplificare, expresia lui P poate fi:

$$P = c \oplus (a \oplus b) = a \oplus (b \oplus c) = b \oplus (c \oplus a).$$

Această observație demonstrează pe de o parte comutativitatea și asociativitatea operației XOR și, pe de altă parte, ne-a permis să folosim o singură poartă XOR pentru a conecta toate cele trei intrări.

Expresia funcției de paritate P este similară cu expresia cifrei sumei s_i de la rangul i pentru circuitul sumator complet.

Dacă numărul de variabile de intrare este impar atunci această definiție a funcției de paritate este echivalentă cu următoarea:

$$P(a,b,c) = \begin{cases} 1, & \text{dacă numărul valorilor de intrare egale cu 1 este impar} \\ 0, & \text{în rest.} \end{cases}$$

Să se arate că dacă prima definiție a funcției de paritate P se modifică și se numără valorile de intrare egale cu 1 atunci funcția rezultat este negarea funcției inițiale.

2.3. Problema 3

Să se construiască un circuit logic care primește pe intrare un număr natural A reprezentat binar pe n poziții și pune pe ieșire reprezentarea binară pe n poziții a valorii $A \div 2$ (câtul împărțirii lui A la 2).

Rezolvare

Numărul n de poziții binare ale registrului de reprezentare determină „puterea” circuitului soluție în sensul următor. Deoarece pe n poziții binare se pot reprezenta valorile naturale zecimale de la 0 la $2^n - 1$, rezultă că, cu cât n este mai mare, cu atât circuitul poate primi pe intrare mai multe valori distincte, deci domeniul de definiție al funcției **div** considerate este mai mare.

Fie $n=3$. Rezultă că vom putea avea pe intrare una din valorile 0, 1, 2, 3, 4, 5, 6, 7. Fie, de asemenea, reprezentarea binară a lui A de forma $a_2a_1a_0$ și reprezentarea binară pentru $A \div 2$ de forma $d_2d_1d_0$.

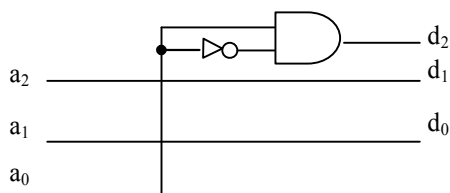
Atunci tabela de valori a funcției este:

A	A div 2	a_2	a_1	a_0	d_2	d_1	d_0
0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0
2	1	0	1	0	0	0	1
3	1	0	1	1	0	0	1
4	2	1	0	0	0	1	0
5	2	1	0	1	0	1	0
6	3	1	1	0	0	1	1
7	3	1	1	1	0	1	1

De aici se deduc cu ușurință expresiile logice ale funcțiilor de ieșire astfel: $d_2 = 0$, $d_1 = a_2$ și $d_0 = a_1$.

Valoarea logică 0 poate fi exprimată convenabil cu una dintre expresiile: $0 = a_0a_0' = a_1a_1' = a_2a_2'$.

Cu acestea, se poate construi circuitul logic cerut:

**Observație**

În această problemă am considerat funcția **div 2** și valorile de ieșire au depins numai de cifrele de rang 0 și 1 ale intrării (valoarea 0 a lui d_2 se putea construi tot cu a_0 sau cu a_1). Dacă se consideră funcția **modulo 2** (restul împărțirii la 2) atunci se va vedea că singura poziție importantă a intrării este bitul cel mai puțin semnificativ, adică cifra binară a_0 .

2.4. Problema 4

Să se construiască un circuit sumator complet folosind două semisumatoare.

Rezolvare

Dacă notăm cu a și b intrările în circuitul semisumator atunci expresiile funcțiilor de ieșire sunt:

$$s1(a, b) = a \oplus b$$

$$T1(a, b) = ab.$$

Analog, dacă intrările în circuitul sumator complet sunt a , b , c atunci ieșirile sunt determinate de expresiile logice:

$$s2(a, b, c) = c \oplus (a \oplus b)$$

$$T2(a, b, c) = ab + c(a \oplus b).$$

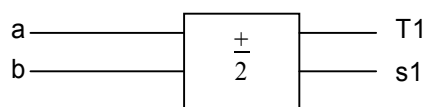
Problema cere să construim ieșirile $s2$ și $T2$ folosind două semisumatoare.

Considerând expresiile anterioare ca notații pentru $s1$ și $T1$ avem

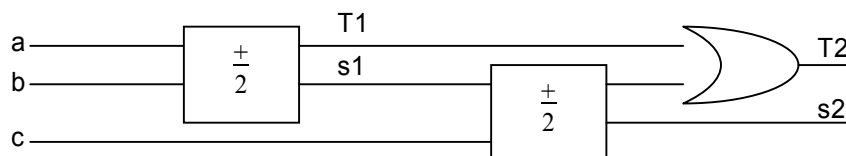
$$s2(a, b, c) = c \oplus s1(a, b) = s1(c, s1(a, b))$$

$$T2(a, b, c) = T1(a, b) + c \cdot s1(a, b) = T1(a, b) + T1(c, s1(a, b)).$$

Fie reprezentarea schematică a circuitului semisumator de forma:



Se observă din expresiile anterioare că a și b sunt intrări în primul circuit semisumator, iar c și $s1(a, b)$ sunt intrările în al doilea circuit semisumator. Atunci reprezentarea finală este:



2.5. Problema 5

Să se construiască un circuit logic combinațional pentru funcția f dată de următoarea tabelă de valori. Atât valoarea de intrare în circuit, cât și valoarea de ieșire se vor reprezenta pe trei poziții binare, adică $IN = xyz_{(2)}$ și $OUT = ABC_{(2)}$.

IN	OUT
0	1
1	2
2	3
3	4
4	3
5	4
6	5
7	6

Rezolvare

Tabela de valori binare pentru funcția f este următoarea:

x	y	z	A	B	C
0	0	0	0	0	1
0	0	1	0	1	0
0	1	0	0	1	1
0	1	1	1	0	0
1	0	0	0	1	1
1	0	1	1	0	0
1	1	0	1	0	1
1	1	1	1	1	0

Conform cu această tabelă de valori avem următoarele expresii logice pentru variabilele de ieșire A, B, C :

$$A = \underline{x'y'z} + xy'z + xyz' + \underline{xyz} = (x' + x)yz + x(y'z + yz') = yz + x(y \oplus z);$$

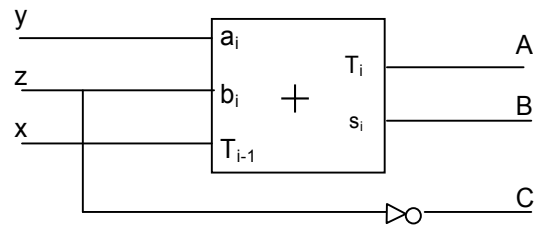
$$B = \underline{x'y'z} + \underline{x'yz'} + xy'z' + xyz = x'(y'z + yz') + x(y'z' + yz) =$$

$$= x'(y \oplus z) + x(y \oplus z)' = x \oplus (y \oplus z);$$

$$C = x'y'z' + x'yz' + xy'z' + xyz' = (x'y' + x'y + xy' + xy)z' = (x' + x)z' = z'.$$

Cea mai simplă reprezentare a acestui circuit folosește un sumator complet, ținând cont de următoarele observații:

- ieșirea A este chiar cifra de transport a circuitului sumator complet;
- ieșirea B este chiar cifra sumei a circuitului sumator complet.



2.6. Problema 6

Să se arate că funcția NAND poate unic generator pentru mulțimea funcțiilor logice elementare.

Rezolvare

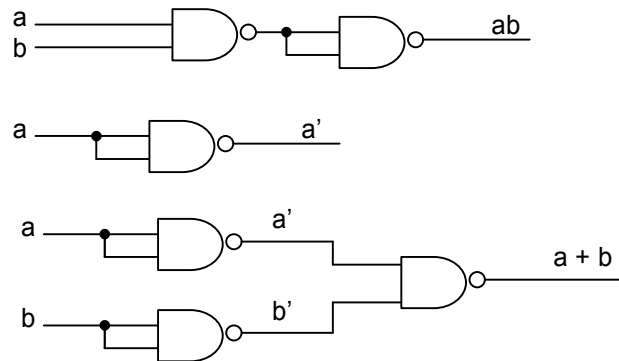
Concret, această problemă cere ca, folosind numai porți NAND, să se construiască circuite logice pentru funcțiile AND, OR, NOT.

Construcția circuitelor se bazează pe următoarele identități:

$$((ab)' \cdot (ab)')' = (ab)'' = ab$$

$$((aa)' \cdot (bb)')' = (a' \cdot b')' = a + b$$

$$(aa)' = a'$$



Observație

Aceeași problemă se poate pune pentru funcția NOR.

NOTIȚE

3. Probleme cu circuite logice combinaționale decodoare, multiplexoare și demultiplexoare

3.1. Problema 1

Să se construiască un circuit decodor folosind numai porți NOR.

Rezolvare

Considerăm cazul unui decodor de tip 2x4, cu intrările notate a și b. Construirea unui circuit decodor poate înlocui porțile AND cu porți NOR, pe baza identităților următoare:

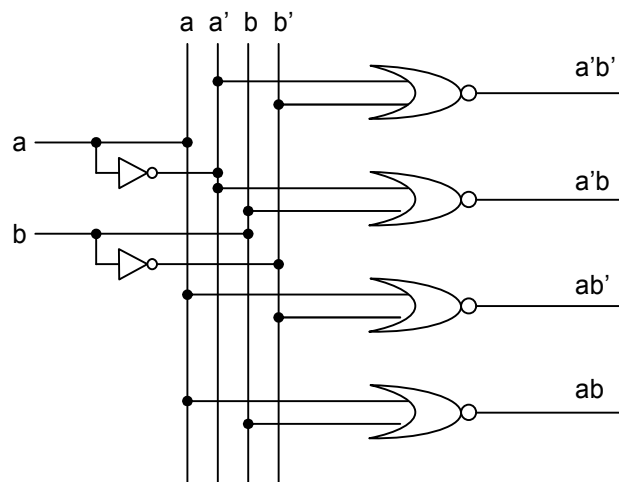
$$a'b' = (a + b)'$$

$$a'b = (a + b')'$$

$$ab' = (a' + b)'$$

$$ab = (a' + b')'$$

Reprezentarea porților logice ale circuitului este:



3.2. Problema 2

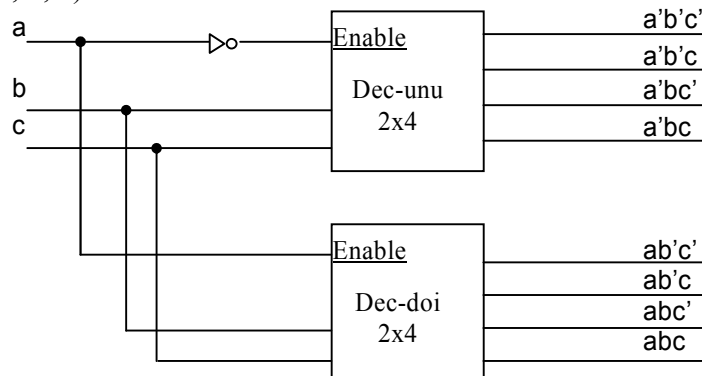
Să se construiască un decodor de tip 3x8 folosind două decodoare de tip 2x4.

Rezolvare

Construcția care se cere în această problemă se bazează pe următoarea observație asupra cuvintelor binare care reprezintă ieșirile din circuitul decodor.

Pentru decodorul de tip 2x4, ieșirile corespund, IN ORDINE, secvențelor binare: 00 – 01 – 10 – 11. Pentru decodorul de tip 3x8 secvențele sunt: 000 – 001 – 010 – 011 – 100 – 101 – 110 – 111, unde am subliniat sufixele date de biții cei mai puțin semnificativi. Comparând secvențele de la decodorul de tip 2x4 cu sufixele subliniate se vede că primele patru sufixe subliniate și ultimile patru sufixe subliniate sunt tocmai secvențele corespunzătoare ieșirilor din decodorul de tip 2x4.

Coroborând această observație cu structura internă de porți logice care formează circuitul decodor, este justificată următoarea construcție – soluție pentru decodorul de tip 3x8 (pentru care notăm intrările cu a, b, c):



Intrarea a a decodului de tip 3x8 este legată la intrări de validare în decodoarele de tip 2x4. Aceasta asigură că, în funcție de valoarea depusă pe intrarea a , numai unul dintre decodoarele de tip 2x4 va funcționa, deci numai unul va depune o valoare 1 pe linii le de ieșire.

Astfel, dacă $a = 0$ atunci numai Dec-unu 2x4 aprinde una dintre ieșiri, în funcție de valorile primite pe intrările de date, adică de pe intrările b și c ale decodului de tip 3x8. Dacă $a = 1$ atunci numai Dec-doi aprinde una dintre ieșirile sale, deoarece pe intrarea de validare Dec-unu 2x4 primește $a' = 0$, deci nu funcționează.

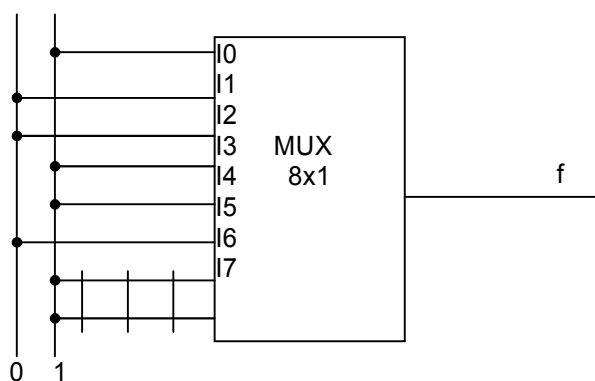
3.3. Problema 3

Să se folosească un multiplexor pentru a reprezenta circuitul logic asociat funcției f dată prin următoarea tabelă de valori binare:

x	y	z	f
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

Rezolvare

Multiplexorul care să reprezinte funcția f dată este:



Multiplexorul din acest circuit preia pe liniile de selecție combinația valorilor de intrare (x, y, z) curente ale funcției f și depune pe linia de ieșire valoarea intrării de date selectată de combinația curentă (x, y, z) .

Justificarea faptului că funcționarea acestui circuit calculează funcția f anterioară rezultă din analiza valorilor puse pe liniile de intrare de date ale multiplexorului, așa cum se vede din reprezentarea grafică și din următorul tabel:

X	y	z	Intrarea selectată	f
0	0	0	I0	1
0	0	1	I1	0
0	1	0	I2	0
0	1	1	I3	1
1	0	0	I4	1
1	0	1	I5	0
1	1	0	I6	1
1	1	1	I7	1

Observație

Dacă se modifică valorile funcției f atunci trebuie modificate corespunzător conexiunile între liniile de tensiune constantă (care reprezintă constant valorile logice 0 sau 1) și liniile de intrare de date ale multiplexorului.

3.4. Problema 4

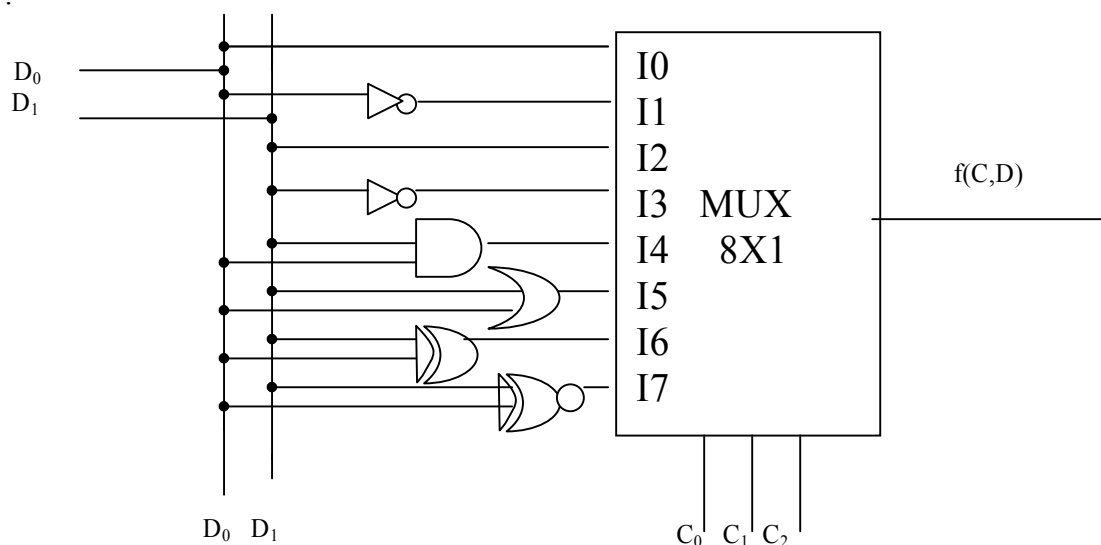
Fie funcția f cu variabilele de intrare $C = (C_0, C_1, C_2)$ și $D = (D_0, D_1)$. Valoarea calculată de funcția f este $y = f(C, D)$ definită prin:

C_0	C_1	C_2	$f(C, D) = y$
0	0	0	D_0
0	0	1	D_0'
0	1	0	D_1
0	1	1	D_1'
1	0	0	$D_0 D_1$
1	0	1	$D_0 + D_1$
1	1	0	$D_0 \oplus D_1$
1	1	1	$(D_0 \oplus D_1)'$

Se cere să se construiască un circuit logic care să depună pe ieșirea y valoarea logică conform cu tabelul de valori alăturat.

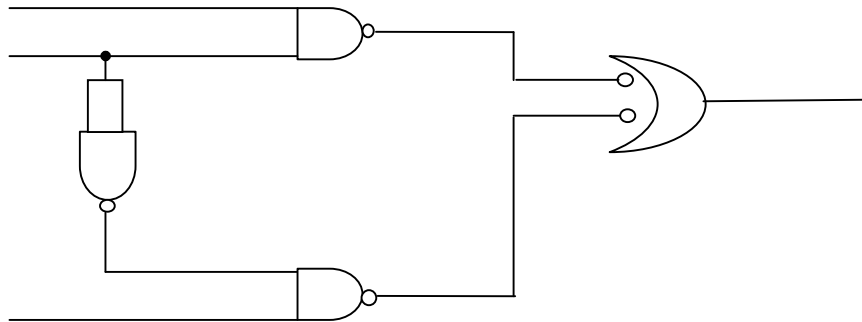
Rezolvare

Multiplexorul care să reprezinte funcția f dată are pe liniile de selecție valorile curente ale variabilelor C_0, C_1, C_2 , iar pe liniile de intrare de date primește valori calculate în funcție de D_0 și D_1 .

**3.5. Problema 5**

Pentru circuitul din reprezentarea următoare, se cere:

- sa se deduca expresia logica a variabilei de iesire Y ;
- sa se simplifice expresia dedusa la a);
- sa se reprezinte circuitul corespunzator expresiei de la b);
- sa se reprezinte circuitul anterior folosind un multilexor.



În reprezentare anterioară fiecare simbol “o” reprezintă o negație.

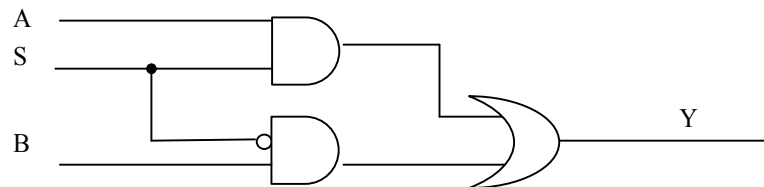
Rezolvare

Expresia logică pentru Y este:

$$Y = ((AS)')' + (((SS)'B)')'$$

Simplificăm expresia anterioară ținând cont de relațiile : $a'' = a$ și $aa = a$. Obținem că $Y = AS + S'B$

Circuitul corespunzător acestei expresii simplificate a lui Y este:



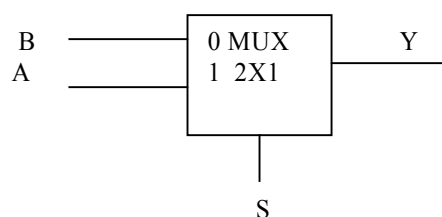
Pentru a justifica utilizarea multiplexorului în reprezentarea acestui circuit vom deduce o altă formă de exprimare a valorii de ieșire Y.

Conform cu expresia $Y = AS + S'B$, avem următoarea tabelă de valori:

S	A	B	Y
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

Din această tabelă de valori se deduce că $Y = \begin{cases} B, & \text{pentru } S = 0 \\ A, & \text{pentru } S = 1 \end{cases}$

Această exprimare este justificată pentru echivalența circuitelor anterioare cu următorul circuit cu multiplexor:



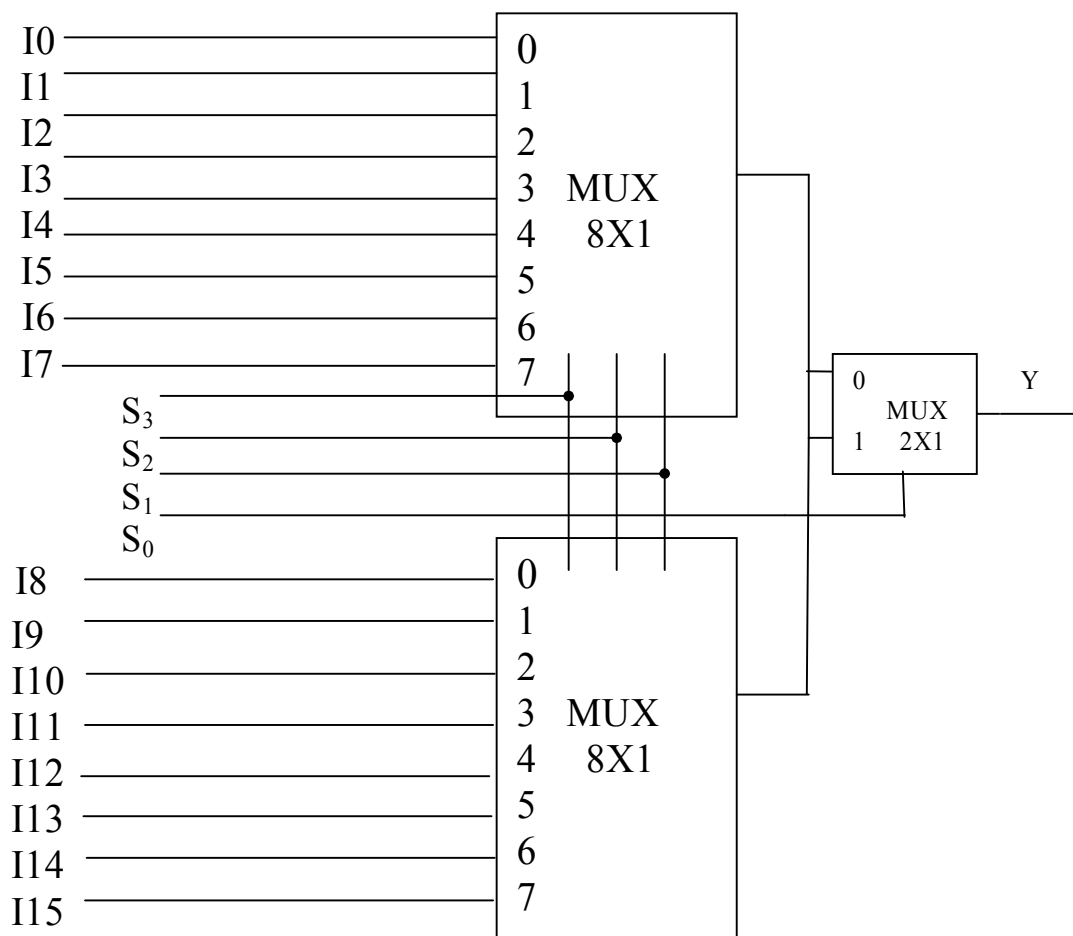
Datorita acestei reprezentari finale, putem spune ca si circuitul initial este unul selector. De aici si denumirea intrarii S- intrare de selectie: in functie de valoarea lui S, Y ia valoarea lui B, sau valoarea lui A. Sa se construiasca un multiplexor de tip 16x1 folosind doua multiplexoare de tip 8x1 si un multiplexor de tip 2x1.

3.6. Problema 6

Sa se construiasca un multiplexor de tip 16x1 folosind doua multiplexoare de tip 8x1 si un multiplexor de tip 2x1.

Rezolvare

Credem ca experienta acumulata pana in acest moment va permite sa intelegeti acest circuit, chiar fara explicatii suplimentare. Intrarile in multiplexorul solutie sunt notate cu: I0, I1, I2, I3, I4, I5, I6, I7, I8, I9, I10, I11, I12, I13, I14, I15. Intrarile de selectie sunt: S0, S1, S2, S3, iar iesirea este Y.



3.7. Problema 7

Sa se construiasca diagrama logica a unui demultiplexor de 2 biti.

Un demultiplexor de tip 1x2n este un circuit logic combinational cu functionare duala multiplexorului: in functie de combinatia de valori de pe cele n linii de selectie, depune pe una din cele 2n linii de iesire valoarea primita pe unica linie de intrare in circuit.

Rezolvare

Schematic, un demultiplexor pe 2 biti se reprezinta astfel:

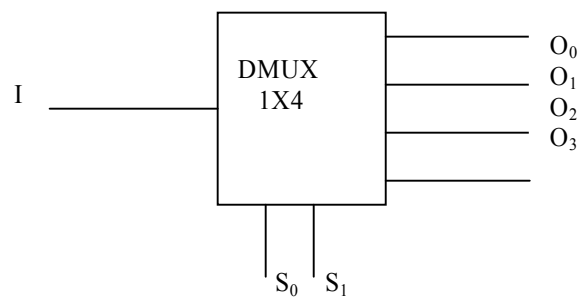
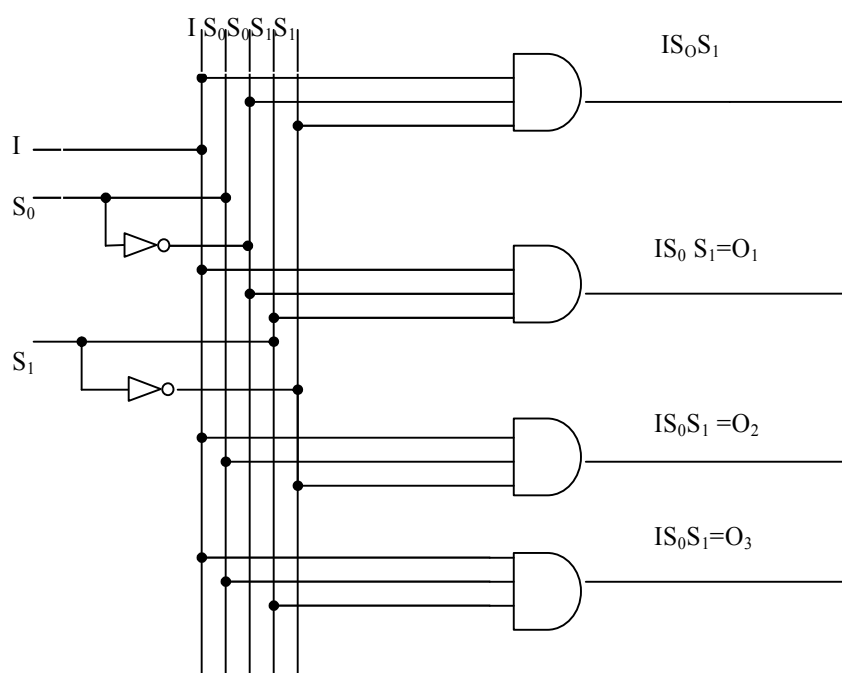


Tabela de functionare determina indicele i al iesirii care va prelua valoarea pusa pe intrarea de date I .

S_0	S_1	OUT
0	0	$I \rightarrow O_0 \quad IS_0'S_1'$
0	1	$I \rightarrow O_1 \quad IS_0'S_1$
1	0	$I \rightarrow O_2 \quad IS_0S_1'$
1	1	$I \rightarrow O_3 \quad IS_0S_1$

Circuitul corespunzator acestei tabele de valori se poate reprezenta astfel:



NOTIȚE

4. Probleme cu circuite logice secventiale bistabile

4.1. Problema 1

Fie un circuit logic secvential cu o intrare, notata x si cu o iesire, notata y . Circuitul consta din doi bistabili de tip D, pentru care intrarile sunt notate D_A , respectiv D_B , iar iesirile cu A , respectiv B .

Ecuatiile care definesc valorile puse pe liniile de circuit sunt:

$$D_A = Ax + Bx$$

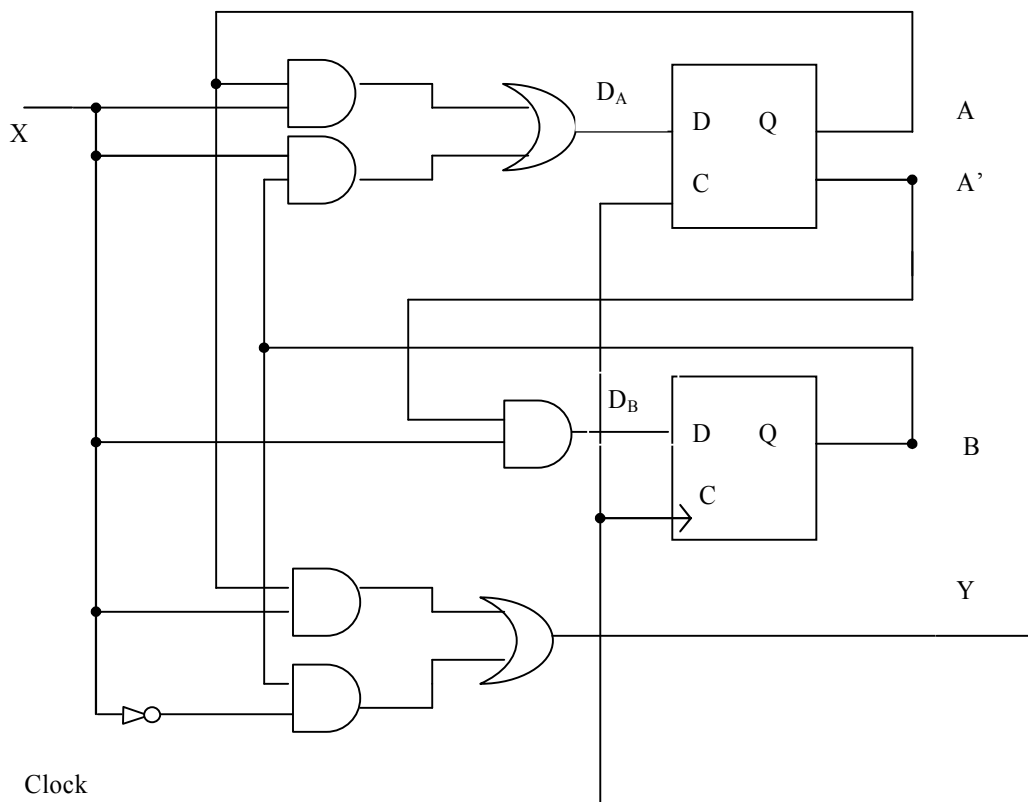
$$D_B = A'x$$

$$y = Ax' + Bx'$$

Sa se reprezinte grafic schema acestui circuit secvential.

Rezolvare

Reprezentarea cu porti a acestui circuit este:



4.2. Problema 2

Fie un circuit secvential cu doi bistabili de tip D, ale caror intrari sunt numite D_A si D_B . Circuitul are doua intrari, x si y si o iesire, z . Ecuatiile intrarilor in bistabili si expresia iesirii z sunt:

$$D_A = x'y + xA$$

$$D_B = x'B + xA$$

$$z = B$$

a) Sa se deseneze diagrama logica a circuitului;

b) Sa se construiasca tabela de valori pentru functia z.

Rezolvare

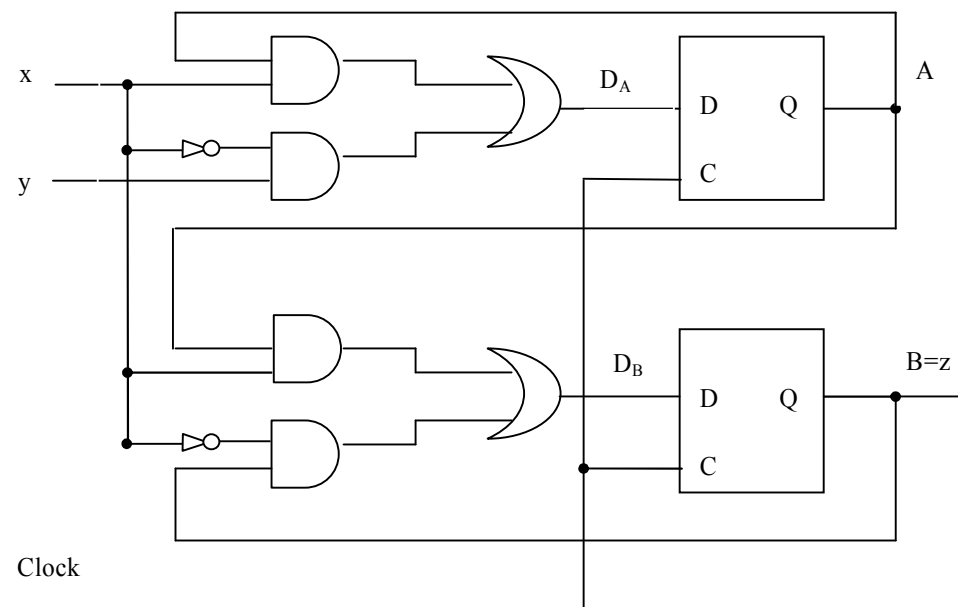
Pentru a deduce tabela de valori asociata circuitului trebuie sa consideram toate perechile de valori binare posibile pe intrarile x si y in circuit, combinate cu toate starile posibile ale bistabililor. Tinand cont de faptul ca circuitul este unul secvential(deoarece are bistabili), rezulta ca este justificata notatia A, B pentru starile curente ale bistabililor si A – urmator, respectiv B – urmator pentru starile urmatoare.

Observatie

In general, pentru deducerea corecta a tabelii de valori asociata unui circuit secvential se va tine seama de urmatoarele reguli:

1. la un acelasi impuls de tact se transmit toate valorile la intrarile in bistabili;
2. la urmatorul impuls de tact, simultan toate valorile de pe intrarile in bistabili trec prin bistabilii respectivi, modificandu-le sau nu starea curenta.

Diagrama logica a circuitului este:



Pentru circuitul anterior, tabela de valori este:

x	y	A	B	A-urmator	B-urmator
0	0	0	0	0	0
0	0	0	1	0	1
0	0	1	0	0	0
0	0	1	1	0	1
0	1	0	0	1	0
0	1	0	1	1	1
0	1	1	0	1	0
0	1	1	1	1	1
1	0	0	0	0	0
1	0	0	1	0	0
1	0	1	0	1	1
1	0	1	1	1	1
1	1	0	0	0	0
1	1	0	1	0	0
1	1	1	0	1	1
1	1	1	1	1	1

Analizand valorile de pe liniile A, B si z se gaseste urmatoarea exprimare pentru aceste variabile de iesire:

$$A\text{-urmator} = \begin{cases} y, & dacax = 0 \\ A, & dacax = 1 \end{cases}$$

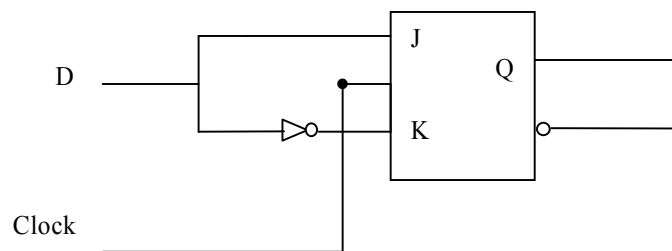
$$Z=B\text{-urmator} = \begin{cases} B, & dacax = 0 \\ A, & dacax = 1 \end{cases}$$

4.3. Problema 3

Sa se reprezinte un circuit basculant bistbil de tip D folosind unul de tip JK.

Rezolvare

Tinand cont de tabelele de functionare pentru bistabilii de tip D si JK rezulta urmatoarea constructie corecta:



NOTIȚE

5. Probleme cu registri

5.1. Problema 1

Continutul initial al unui registru de deplasare este 11011011 (registru este pe 8 biti). Se aplica de sase ori deplasarea la dreapta cu intrarea seriala 101101. Care este continutul registrului respectiv dupa fiecare deplasare?

Rezolvare

11011011 este continutul initial al registrului;

11011011 este continutul dupa prima deplasare;

0110110 este continutul dupa a doua deplasare;

10111011 este continutul dupa a treia deplasare;

11011101 este continutul dupa a patra deplasare;

01101110 este continutul dupa a cincea deplasare;

10110111 este continutul dupa ultima deplasare;

In reprezentarea continutului registrului la fiecare pas am folosit urmatoarele conventii:

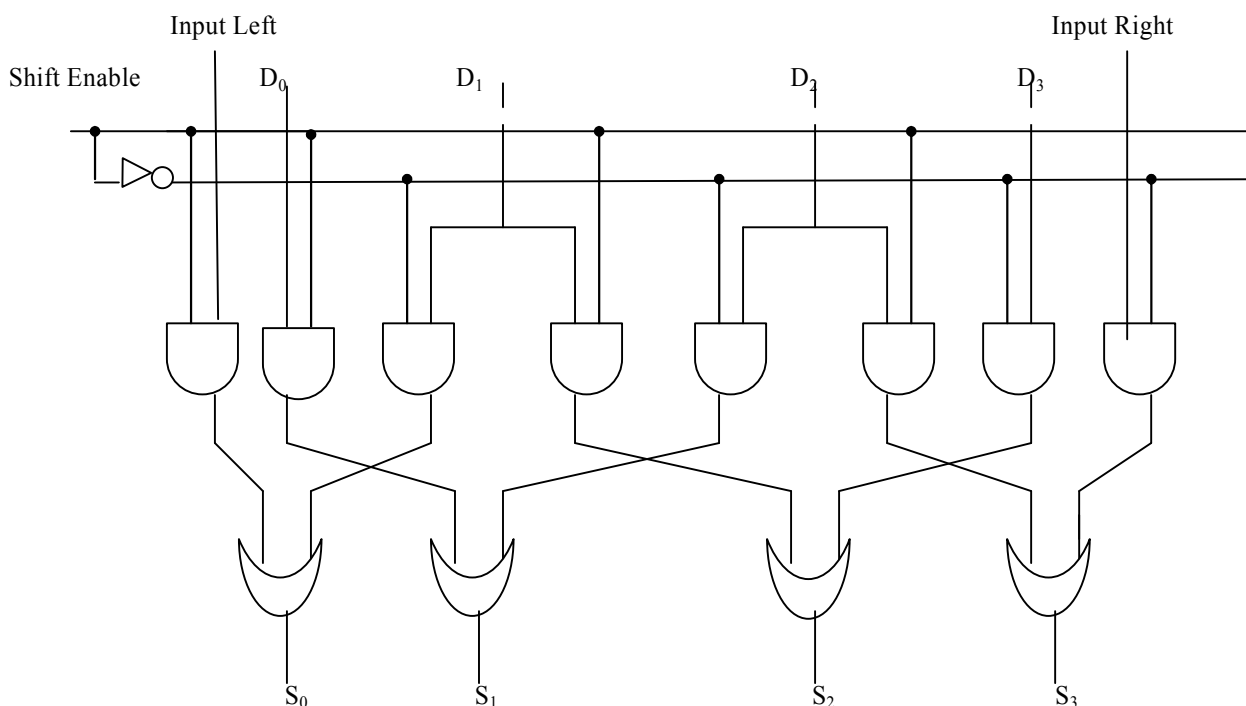
- bitul care intra pe intrarea seriala la fiecare pas este reprezentat cu caracter inclinat;
- bitii care se deplaseaza, dar au intrat de pe intrarea principala la pasii anteriori sunt reprezentati cu caractere subliniate;
- bitii care se deplaseaza si fac parte din continutul initial al registrului sunt reprezentati cu caractere ingrosate

Observatie

Aceiasi problema se poate pune considerand ca registrul executa o microoperatie de deplasare la stanga. Care este in acest caz continutul final al registrului dupa sase deplasari?

5.2. Problema 2

Sa se analizeze functionarea circuitului urmator si sa se justifice faptul ca este un circuit de deplasare.



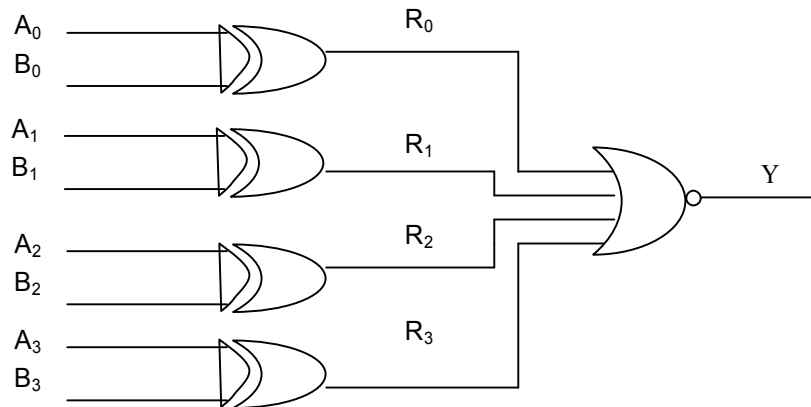
Rezolvare

Acest circuit functioneaza conform cu urmatoarele reguli:

- 1) daca ShiftEnable = 0 atunci pe iesirile S_0, S_1, S_2, S_3 avem respectiv, valorile de pe intrarile: $D_1, D_2, D_3, \text{InputRight}$. Se observa de aici ca circuitul functioneaza cu deplasare la dreapta.
- 2) daca ShiftEnable = 1 atunci pe iesirile S_0, S_1, S_2, S_3 avem respectiv, valorile de pe intrarile: $\text{InputLeft}, D_1, D_2, D_3$. Se observa de aici ca circuitul functioneaza cu deplasare la stanga.

5.3. Problema 3

Sa se deduca rolul urmatorului circuit:

**Rezolvare**

Pentru fiecare $i \in \{0, 1, 2, 3\}$ tabela de valori pentru linia $R_i = A_i \oplus B_i$ este:

A_i	B_i	$A_i \oplus B_i$
0	0	0
0	1	1
1	0	1
1	1	0

De aici putem da urmatoarea interpretare pentru valorile lui R_i

$$R_i = \begin{cases} 0, & A_i = B_i \\ 1, & A_i \neq B_i \end{cases}$$

Valoarea Y este data de $Y = \text{NOR}(R_0, R_1, R_2, R_3)$, deci

$$Y = \begin{cases} 1, & \text{cand } R_0 = R_1 = R_2 = R_3 = 0 \\ 0, & \text{in rest} \end{cases}$$

Coroborand exprimarile pentru $R_i, i \in \{0, 1, 2, 3\}$ si pentru Y, avem ca

$$Y = \begin{cases} 1, & \text{cand } A \equiv B \\ 0, & \text{cand } A \neq B \end{cases}$$

unde, prin $A \equiv B$ am inteles ca $(\forall i \in \{0, 1, 2, 3\}) [A_i = B_i]$, adica continutul registrului A este identic cu continutul registrului B.

In concluzie, putem spune ca circuitul propus este un **circuit comparator pe 4 biti** deoarece compara continutul a doi registri A si B, fiecare pe 4 biti si transmite la iesire valoarea 1 daca continuturile sunt identice si valoarea 0 daca continuturile difera cel putin printr-o pozitie.

5.4. Problema 4

Pornind de la reprezentarea logica a registrului de numarare binara prezentat la curs, sa se construiasca un registru pentru microoperatia de numarare binara inversa.

Rezolvare

Vom construi aceasta numaratoare pe 2 biti.

Pentru a deduce structura logica a registrului de numarare inversa trebuie sa determinam mai intai starile succesive ale registrului. Daca starea initiala este 00 atunci starile urmatoare vor fi: 11, 10, 01, 00, 11, 10, 01, s.a.m.d.

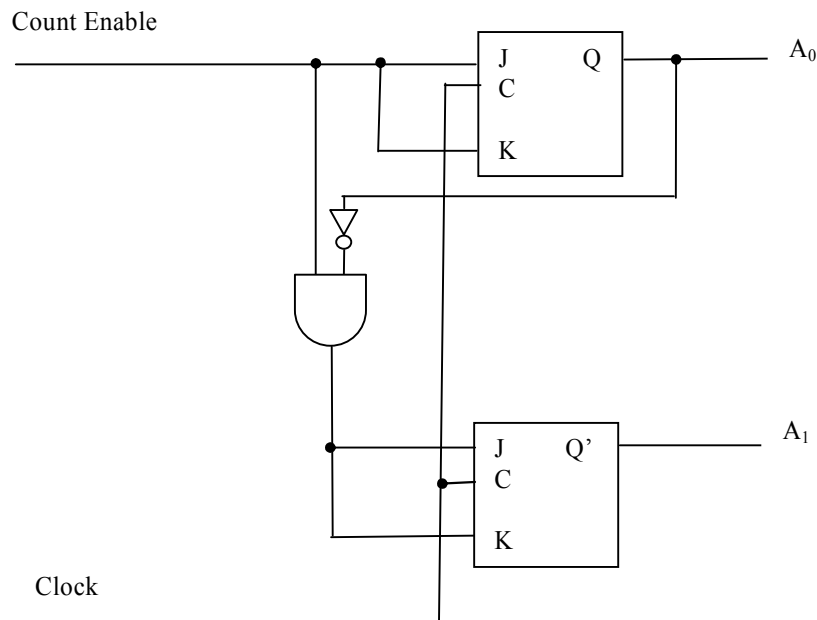
Se observa ca aceste stari se repeta cu o periodicitate de patru stari: 00, 11, 10, 01. Regulile care se pot stabili pentru determinarea unica a starii urmatoare pornind de la starea curenta sunt:

- 1) bitul cel mai putin semnificativ se complementeaza intotdeauna;
- 2) oricare dintre ceilalti biti se complementeaza numai daca toti bitii de rang mai mic sunt 0.

Observatie

Aceste reguli sunt analoage celor de la numaratoarea directa, cu deosebirea la regula a doua ca bitul se complementeaza daca are numai 0 la dreapta si nu numai 1, ca la numaratoarea directa.

Pe baza acestor reguli si a observatiei anterioare care da asemanarea cu registrul de numarare directa, se poate construi circuitul logic folosind tot circuite basculante bistabile de tip JK:

**NOTIȚE**

6. Probleme recapitulative

Problemele propuse in acest capitol au un grad mai mare de dificultate. O astfel de problema poate folosi cunostintele din mai multe dintre capitolele anterioare.

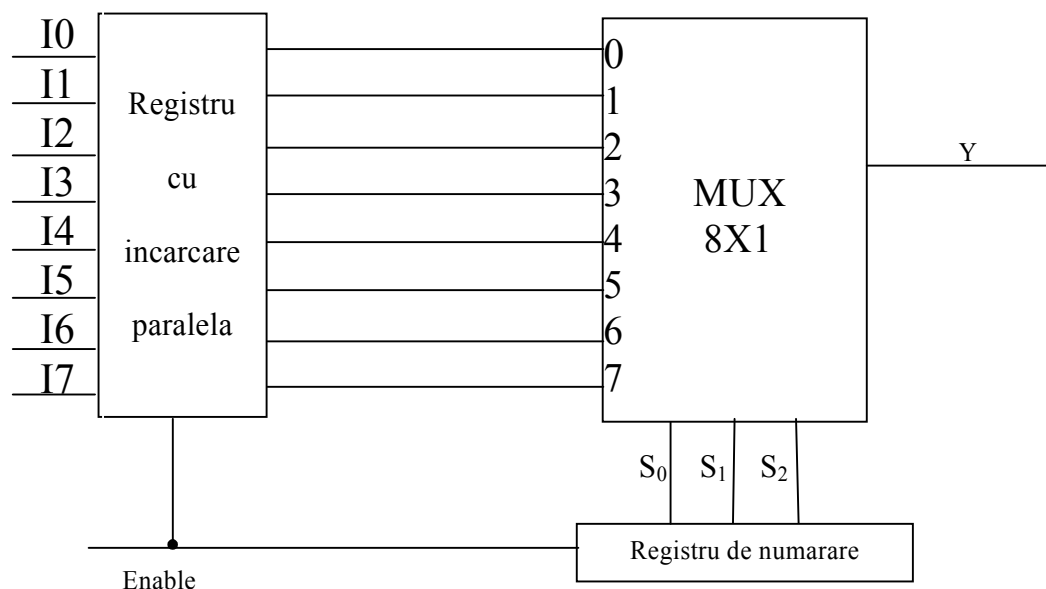
6.1. Problema 1

Sa se construiasca un convertor paralel – serie.

Rezolvare

Ne propunem sa construim un astfel de circuit pe 8 biti. Un circuit convertor paralel – serie primeste simultan (in paralel) pe 8 lini de intrare de date valorile de intrare, pe care le depune succesiv (in serie) pe unica iesire din circuit.

Un model imediat pentru un astfel de circuit foloseste un multiplexor de tip 8x1, pentru care, configuratiile succesive ale valorilor de pe intrarile de selectie trebuie sa asigure ca MUX depune pe iesire, in ordinea I_0, I_1, \dots, I_7 , valorile primite pe intrarile de date. Pentru a asigura aceste configuratii succesive pe intrarile de selectie, acestea isi vor prelua valorile dintr-un registru de numarare de trei biti.



Starea initiala a registrului de numarare trebuie sa fie 000 pentru ca prima valoare depusa de MUX pe iesire sa fie ceea ce primeste pe intrarea I_0 . Registrul de numarare va trece prin toate stările distincte succesive, pana la starea 111, care corespunde momentului in care MUX depune pe iesire valoarea primita pe intrarea I_7 .

Valoarea de pe intrarea de validare Enable a functionarii circuitului va asigura ca:

- ✓ la un anumit impuls de tact se incarca simultan (in paralel) valori pe liniile de date I_0, I_1, \dots, I_7 ;
- ✓ se initializeaza starea registrului de numarare cu configuratia 000;
- ✓ atat timp cat registrul de numarare parcurge un ciclu de stari distincte (de la 000 la 111) nu se incarca alte valori pe intrarile I ale multiplexorului.

In acest fel, MUX asigura depunerea pe iesirea Y a intrarilor I_0, I_1, \dots, I_7 bit dupa bit.

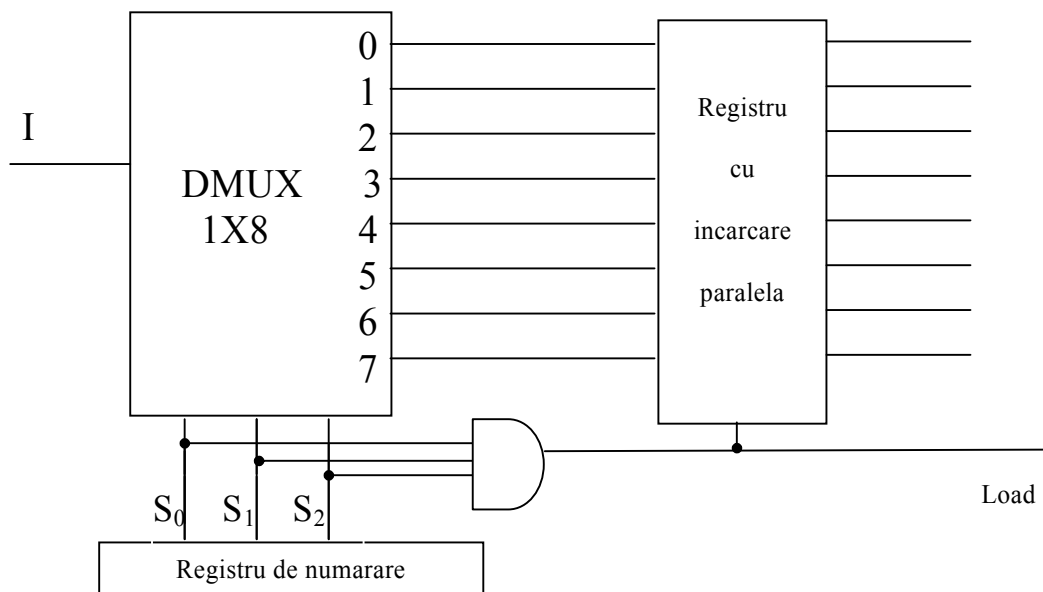
6.2. Problema 2

Sa se construiasca un convertor serie – paralel.

Rezolvare

De data aceasta se pune problema inversa fata de cazul din problema anterioara. De aceea, este de asteptat ca circuitul corespunzator unui convertor serie – paralel sa foloseasca un

demultiplexor. În plus, trebuie asigurată depunerea simultană (în paralel) a valorilor pe liniile de ieșire din circuit. Aceasta funcțiune este preluată de poarta AND legată la intrarea de validare Load a circuitului cu încărcare paralelă.



6.3. Problema 3

Să se construiască un circuit logic secvențial cu două intrări notate E și x și care este format din două CBB-JK, ale căror ieșiri sunt notate A și B . Valorile curente ale ieșirilor A și B determină la momentul respectiv starea circuitului.

Dacă la momentul curent starea circuitului este notată Q_t atunci starea următoare, Q_{t+1} , rezultă din următoarea schemă de funcționare a circuitului:

- 1) dacă $E = 0$ atunci $Q_{t+1} = Q_t$;
- 2) dacă $E = 1$ și $x = 1$ atunci circuitul funcționează ca număratoare directă;
- 3) dacă $E = 1$ și $x = 0$ atunci circuitul funcționează ca număratoare inversă.

Rezolvare

Dacă A și B sunt valorile curente ale ieșirilor din bistabili de tip JK atunci notăm cu A^* și B^* respectiv valorile de pe ieșiri la momentul următor.

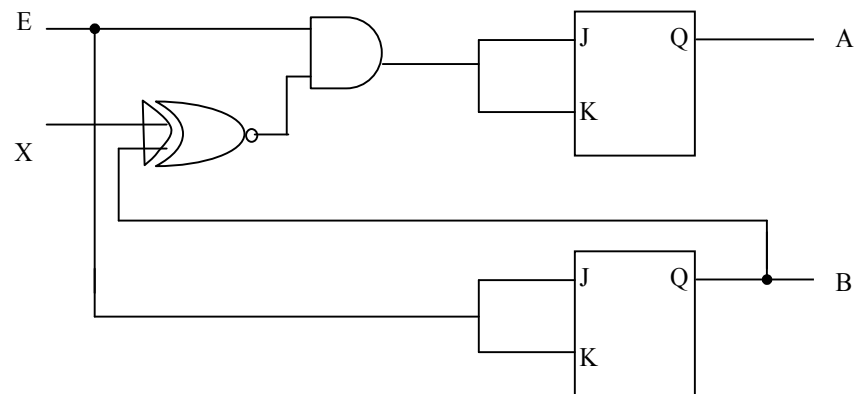
Deci $Q_t \equiv (A, B)$ și $Q_{t+1} \equiv (A^*, B^*)$.

Conform cu tabela de valori, vom reprezenta circuitul pornind de la schemele de reprezentare pentru registrele de numărare.

Tabela de valori pentru funcționarea acestui circuit este următoarea:

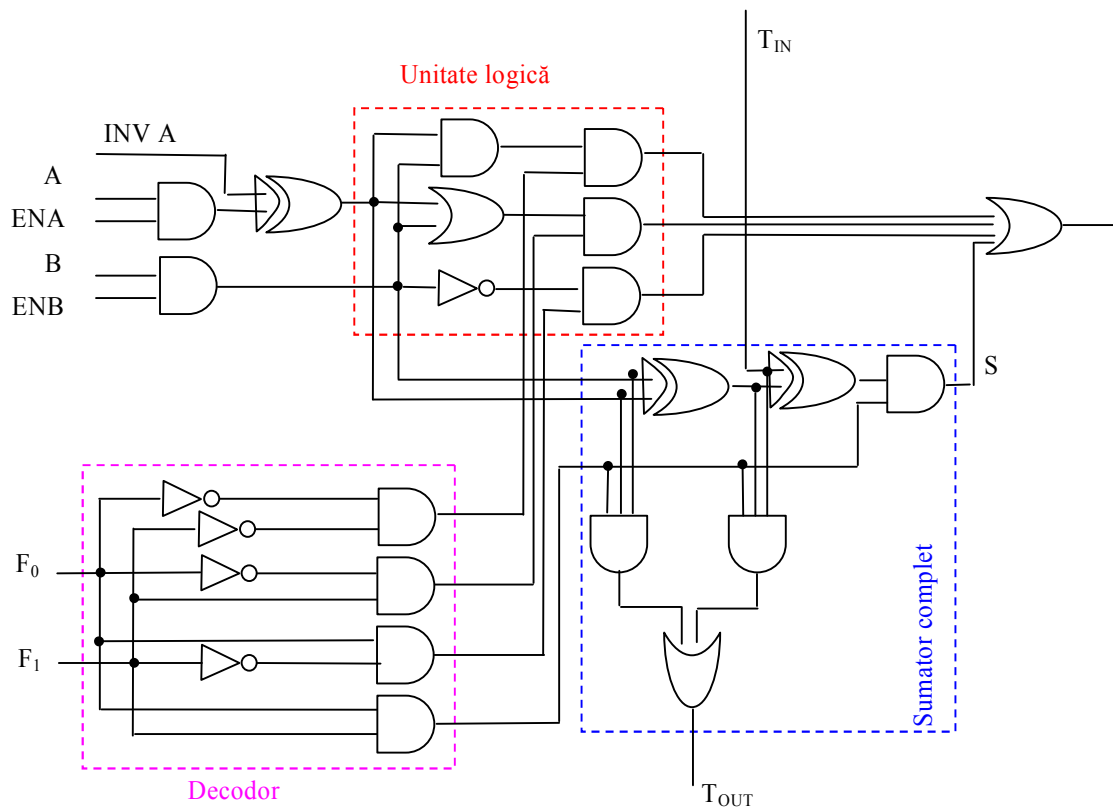
E	x	A	B	A^*	B^*
0	0	0	0	0	0
...
...
0	1	1	1	1	1
1	0	0	0	1	1
1	0	0	1	0	0
1	0	1	0	0	1
1	0	1	1	1	0
1	1	0	0	0	1
1	1	0	1	1	0
1	1	1	0	1	1
1	1	1	1	0	0

Reprezentarea circuitului este:



6.4. Unitate aritmetico-logică pe 1 bit

Prezentăm în continuare modelul unei UAL cu operanzii A și B reprezentați pe 1 bit.



Semnificația intrărilor:

- ✓ intrări de date: A, B;
- ✓ intrări de selecție: F₀, F₁;
- ✓ intrări de validare: ENA, ENB;
- ✓ INVA – intrare de inversare pentru valoarea de pe intrarea A;

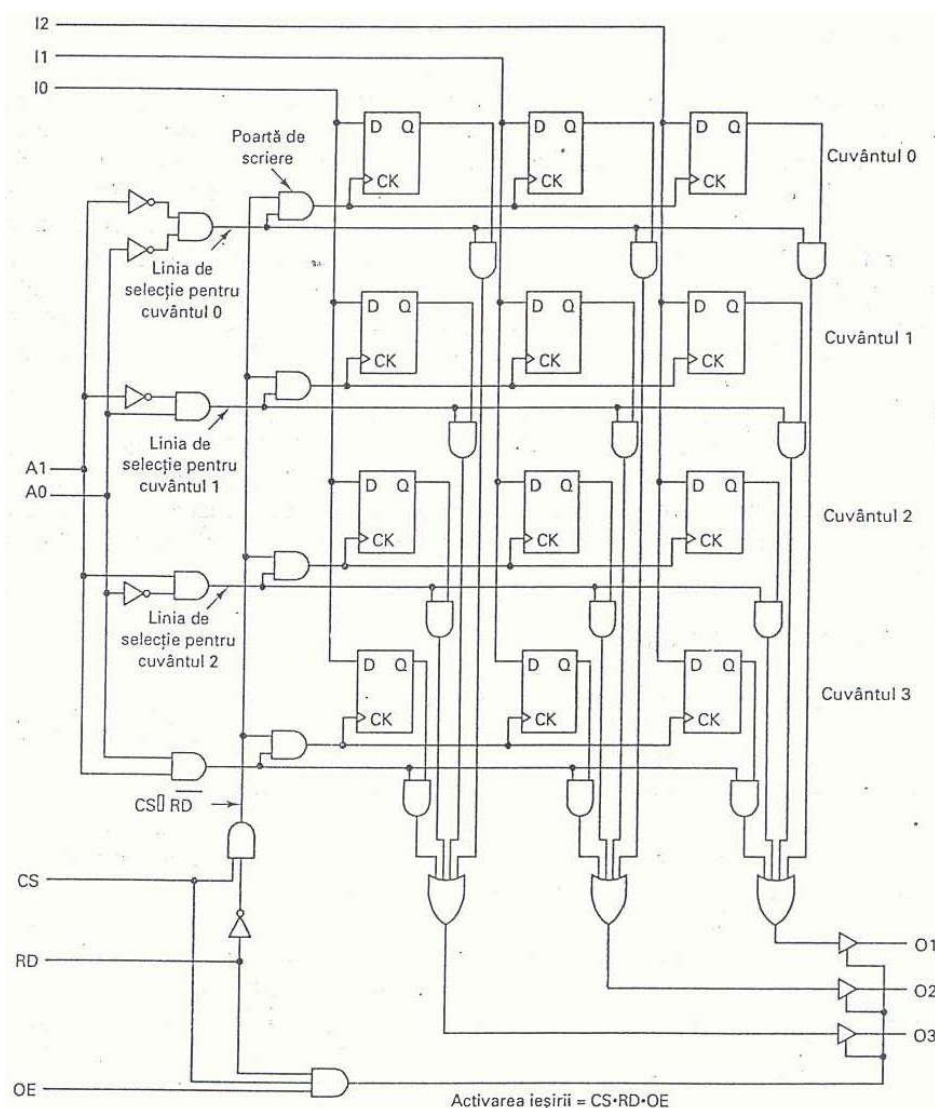
Operația de scădere se efectuează ca adunarea descăzutului cu opusul scăzătorului, unde prin *opus* ne referim la complementul față de 2 (se complementează toți biții și se adună 1 la rezultat).

Înțelegerea circuitului se poate testa prin completarea corectă a valorilor de ieșire pentru diferite combinații de intrare, într-un tabel cu structura următoare:

F0	F1	ENA	ENB	INVA	INC	OUT
0	1	1	0	0	0	A
0	1	0	1	0	0	B
0	1	1	0	1	0	$(A \oplus 1) + 0 = A'$
1	0	0	1	0	0	B'
1	1	1	1	0	0	
...						
1	1	1	0	1	1	$(A \oplus 1) +_2 0 +_2 1 = A' +_2 1 = -A$
0	0	1	1	0	0	
...						

6.5. Memorie de tip 4x3

Următorul circuit reprezintă o structură de memorie cu 4 cuvinte pe 3 biți fiecare.



Semnificația liniilor de circuit:

- ✓ Intrări de date: I0, I1, I2 – valorile sunt folosite de funcția WRITE;
- ✓ Intrări de selecție a cuvântului de memorie: A0, A1;
- ✓ Intrări de validare: CS (*engl.*, Chip Select)
RD (*engl.*, Read)
OE (*engl.*, Output Enable)
- ✓ Ieșiri: O0, O1, O2 – valorile sunt returnate de funcția READ

Funcționarea acestui circuit de memorare se sintetizează în următoarele tabele de valori:

A0	A1	
0	0	Select cuvânt 0
0	1	Select cuvânt 1
1	0	Select cuvânt 2
1	1	Select cuvânt 3

CS	
0	NU
1	DA

RD	
0	WRITE
1	READ

NOTIȚE

7. Modelarea funcționării procesorului

7.1. Mașina cu trei adrese

Pentru a vedea cum funcționează mașina cu trei **adrese** vom construi programul pentru rezolvarea ecuației de gradul al doilea.

În **zona de date** trebuie trecute următoarele elemente:

- ✓ coeficienții a, b, c ai ecuației de rezolvat;
- ✓ constantele 0 și 2;
- ✓ rezultatele intermediare $b^2, 2*a, -b$;

Pentru acestea vom rezerva locații de memorie denumite astfel:

A	B	C	DOI	ZERO	T1	T2	T3	T4
a	b	c	2	0	b^2	$2*a$	$-b$	

Cuvântul T4 este rezervat pentru alte rezultate intermediare necesare programului.

Reprezentarea programului se face astfel: pe prima coloană vom trece numele instrucțiunii de executat și pe a doua coloană – operația; urmează trei coloane pentru adrese și ultima coloană pentru comentarii.

	CIT			A	Citește coeficientul a
	CIT			B	Citește coeficientul b
	CIT			C	Citește coeficientul c
	CIT			DOI	Citește constanta 2
	–	DOI	DOI	ZERO	Construiește constanta 0 ($0=2-2$)
	*	B	B	T1	$T1:=b^2$
	*	DOI	A	T2	$T2:=2*a$
	*	DOI	T2	T3	$T3:=4*a$
	*	T3	C	T3	$T3:=4*a*c$
	–	T1	T3	T1	$T1:=\Delta$
	–	ZERO	B	T3	$T3:=-b$
	>	T1	ZERO	E1	Salt la E1 dacă $\Delta>0$
	=	T1	ZERO	E2	Salt la E2 dacă $\Delta=0$
	/	T3	T2	T3	Cazul $\Delta<0$: $T3:=-b/(2*a)$
	–	ZERO	T1	T1	$T1:=-\Delta$
	$\sqrt{}$	T1		T1	$T1:=\sqrt{-\Delta}$
	/	T1	T2	T1	$T1:=\sqrt{-\Delta}/(2a)$
	TIP			T3	Tipărește partea reală
	TIP			T1	Tipărește partea imaginară
	STOP				
E2:	/	T3	T2	T3	$T3:=-b/(2a)$
	TIP			T3	Tipărește rădăcina comună
	STOP				
E1:	$\sqrt{}$	T1		T1	$T1:=\sqrt{\Delta}$
	–	T3	T1	T4	$T4:=T3-T1$
	/	T4	T2	T4	$T4:=x_1$
	TIP			T4	Tipărește x_1
	+	T3	T1	T4	$T4:=T3+T1$
	/	T4	T2	T4	$T4:=x_2$
	TIP			T4	Tipărește x_2
	STOP				

7.2. Mașina cu o adresă

Vom construi programul pentru mașina cu o adresă care să rezolve ecuația de gradul al doilea.

Zona program începe cu locația de adresă 0 și se termină cu locația de adresă 83. Zona de date ocupă 6 locații, de la 84 la 89, astfel: primele trei locații (LA, LB, LC) rețin coeficienții ecuației, iar următoarele trei (T1, T2, T3) sunt folosite pentru memorarea unor rezultate intermediare.

Vom descrie programul în paralel, atât folosind codurile instrucțiunilor cât și folosind numerele de ordine ale instrucțiunilor de executat.

	CIT	LA	Citirea coeficienților	0:	26	84
	CIT	LB		2:	26	85
	CIT	LC		4:	26	86
	←	LA	$T3 := 2*a$	6:	1	84
	*	2		8:	13	2
	→	T3		10:	2	87
	←	0	$T1 := -b/(2*a)$	12:	1	0
	-	LB		14:	12	85
	/	T3		16:	14	89
	→	T1	$T2 := 4*a*c$	18:	2	87
	←	T3		20:	1	89
	*	2		22:	13	2
	*	LC	$A := b^2 - 4*a*c$	24:	13	86
	→	T2		26:	2	88
	←	LB		28:	1	84
	*	LB	$A := (b^2 - 4*a*c) / [(2*a)^2]$	30:	13	85
	-	T2		32:	12	88
	/	T3		34:	14	89
	/	T3	$Da, \Delta > 0$	36:	14	89
	>	E1		38:	21	62
	=	E2		40:	23	56
	TIP	T1	Tipărește partea reală Anulează val din A	42:	27	87
	←	0		44:	1	0
	-	T2		46:	12	88
	→	T2	$A := \sqrt{-\Delta}$	48:	2	88
	√	T2		50:	15	88
	→	T2		52:	2	88
	TIP	T2	Tipărește partea imaginară	54:	27	88
	STOP			56:	28	0
E2:	TIP	T1	Tipărește soluția unică	58:	27	87
	STOP			60:	28	0
E1:	√	T2		62:	15	88
	→	T2	$x_1 = (-b - \sqrt{\Delta}) / (2*a)$	64:	2	88
	←	T1		66:	1	87
	-	T2		68:	12	88
	→	T3		70:	2	89
	TIP	T3		72:	27	89

	←	T1		74:	1	87
	+	T2		76:	11	88
	→	T3		78:	2	89
	TIP	T3	Tipărește $x_2 = (-b + \sqrt{\Delta}) / (2*a)$	80:	27	89
	STOP			82:	28	0
LA:			a	84:		
LB:			b	85:		
LC:			c	86:		
T1:			$-b / (2*a)$	87:		
T2:			$b^2 - 4*a*c$	88:		
T3:				89:		

În acest program s-au folosit unele modalități de a reduce numărul de operații intermediare. Astfel, valorile unor subexpresii sunt memorate în locații temporare (T1 sau T2) și preluate de mai multe ori.

De asemenea, se poate observa că valoarea conținută în registrul acumulator A este preluată de instrucțiunea care urmează *în timp* (din punct de vedere logic, conform secvenței de instrucțiuni în execuție), chiar dacă aceasta nu este următoarea instrucțiune scrisă în program.