# SISTEM DE CALCUL IN TIMP REAL

SCTR

-SZOKE ENIKO -

Curs 6

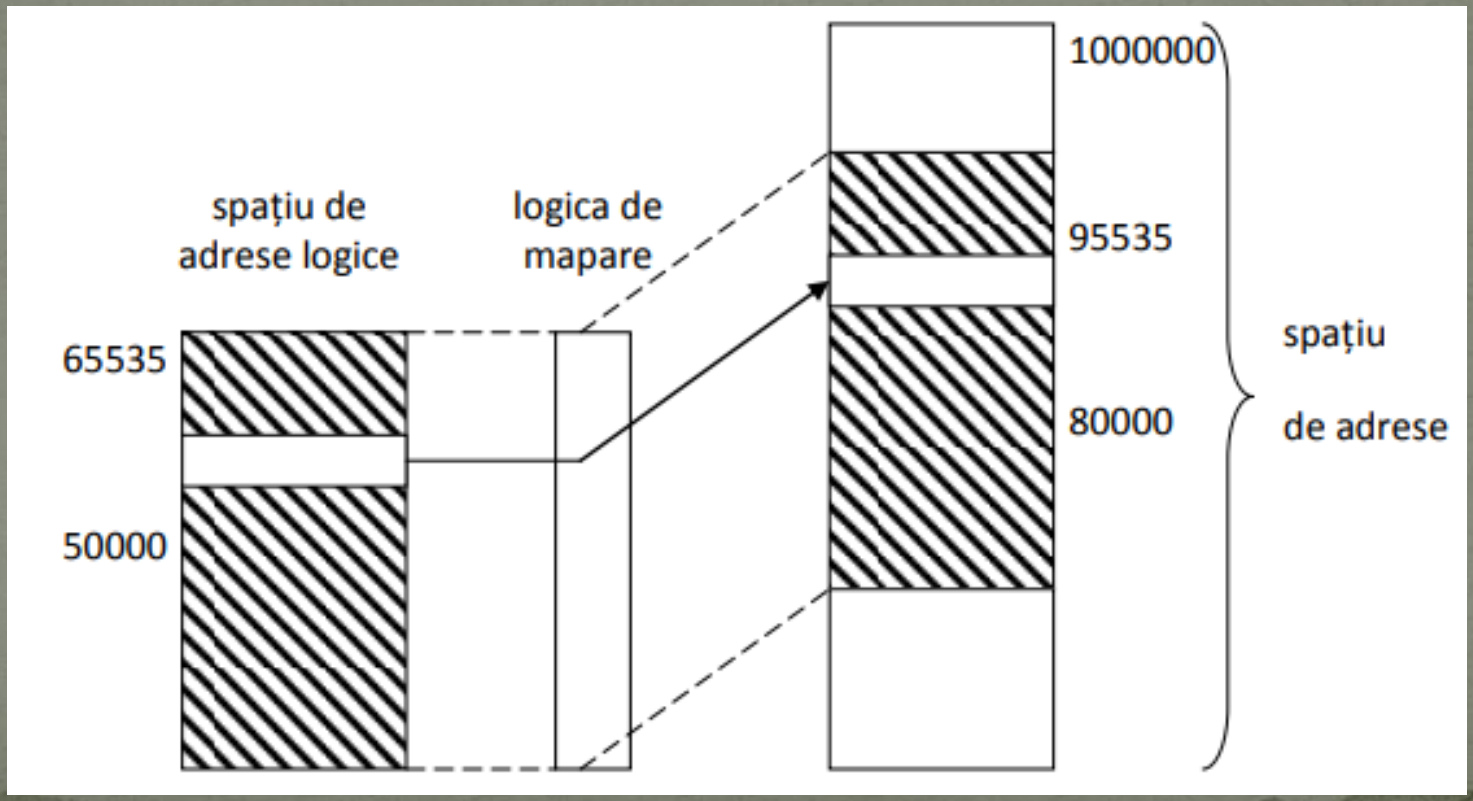
**Memoria virtuală**

Spațiul de adrese logice este mult mai mare decât memoria fizică. Memoria virtuală este un mecanism pentru a extinde limitele memoriei fizice. Într-un sistem cu memorie virtuală, aceasta apare utilizatorului ca și cum întregul spațiu logic de adrese este disponibil pentru memorare. Dar, de fapt, doar câteva pagini din spațiul logic de adrese sunt mapate peste spațiul fizic la un moment dat. Alte pagini nu sunt prezente în memoria principală; în schimb, informația din aceste pagini este memorată într-o memorie secundară, cum ar fi discul, al cărei cost pe bit este mult mai economic.

Ori de câte ori se obține acces la o pagină care lipsește, softul sistemului de operare încarcă pagina respectivă de pe disc și memorează pe disc o altă pagină, la care nu s-a făcut recent referire. Utilizatorul are impresia unei memorii fizice uriașe, dar mai lente.

# Maparea memoriei

În figura următoare se prezintă o operație de mapare foarte simplă. Întregul spațiu de adrese logice ale unui program (locațiile 0-65535, în acest caz) este mapat (suprapus) peste locațiile fizice 30000-95535. Astfel, o referință a unui program la locația 50000, va prelua datele, efectiv de la locația fizică 80000 (30000+50000).



Această operație este foarte utilă în sistemele multiprogram (sau multitask), pentru care a fost dezvoltată maparea. Astfel, fiecare program are propriul său spațiu logic de adrese, care este complet independent de spațiul oricărui alt program. Ca atare, mai multe programe pot partaja memoria fizică fără posibilitatea de a interfera între ele. Procesul de mapare potrivește (suprapune) tot spațiul de adrese logice într-o zonă de memorie fizică, dar procesul este transparent pentru program.

# Unitati de intrare si iesire

- Informatiile (date sau instructiuni) circula pe caile de sistem format din busul de date si busul de adrese si sunt gestionate de unitatile de intrare/iesire.

- Gestionarea marimilor de intrare/iesire:
  - Prin intrari/iesiri programabile
  - Prin intrari/iesiri mapate in memorie
  - Prin metoda de acces direct la memorie sau DMA (Direct Memory Acces)

## Porturi I/E

Prin port se înțelege un loc (în general un registru), cu adresă specifică.

Adresele porturilor de intrare-ieşire pot fi tratate în două moduri:
- ca porturi cu adrese distincte față de adresele de memorie
- adresele porturilor sunt înglobate în spațiul de adrese al memoriei principale. Această organizare a adreselor (numita şi "mapare a adreselor de I/O în spațiul de memorie") face ca porturile să fie selectate prin aceleaşi semnale de adresă şi control ca şi memoria

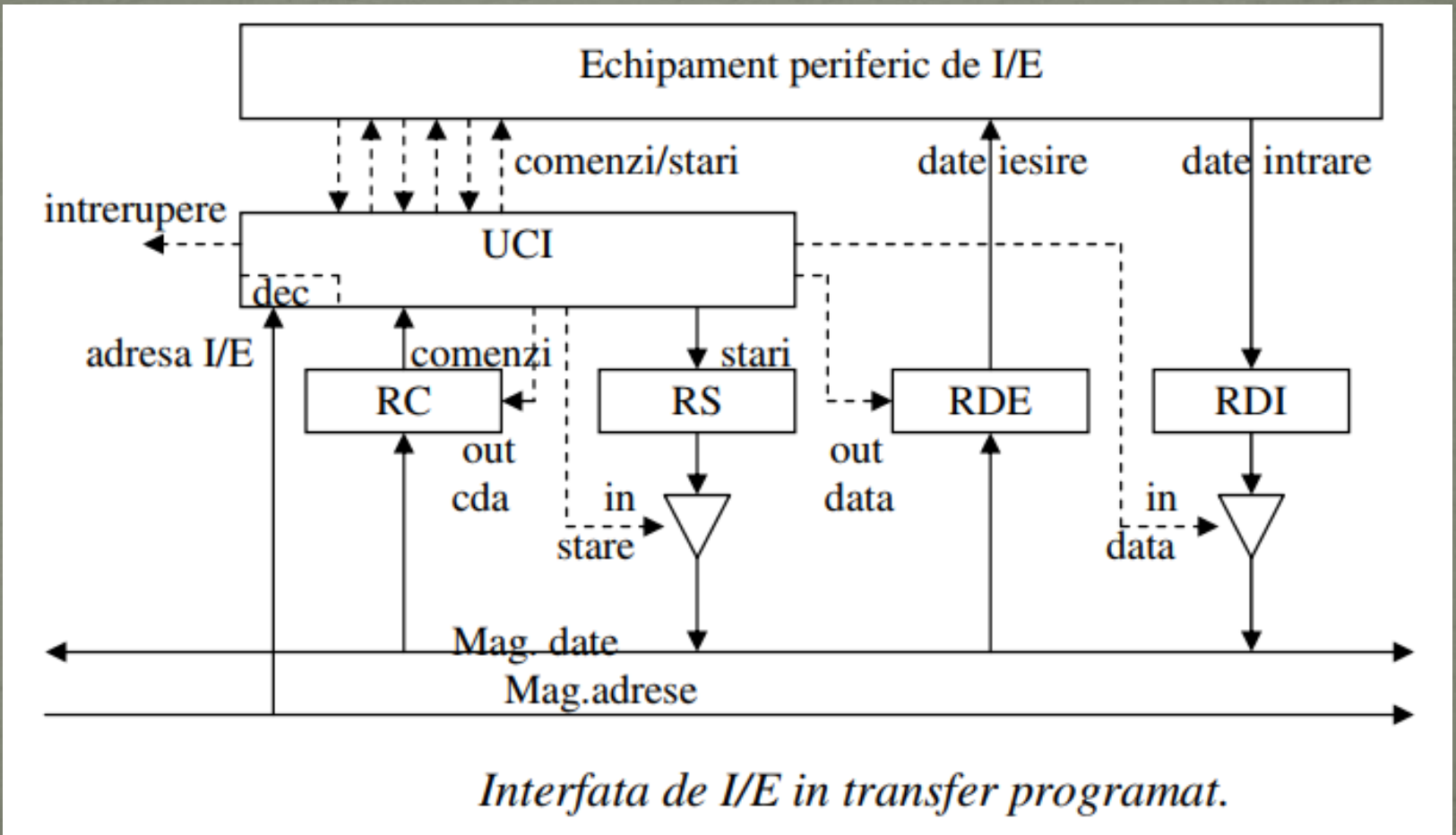
# Unitati de intrare si iesire

- **In modul de lucru cu intrari/iesiri programabile** se utilizeaza instructiuni speciale din setul de instructiuni al procesorului pentru a transfera informatii intre CPU si periferice sau CPU si memorii.

- IN, OUT - in care dispozitivul periferic este recunoscut ca o adresa de catre procesor iar in CPU adresa este a unui registru implicit.

- Acest mod de lucru utilizeaza capacitatile CPU pentru toate operatiunile de I/E si deci se mareste timpul de raspuns al SC  probleme la unele aplicatii in TR

- Acesta este cel mai simplu mod de transfer, cu un cost redus dar si cu performante scazute.
- In cadrul transferului programat exista doua submoduri:
  - cu citirea ciclica a starii
  - in intreruperi




*Interfata de I/E in transfer programat.*

registrul de **control**, care primește comenzi pentru periferic
registrul de **stare**, care conține starea interna a perifericului
registrul de **intrare**, din care se preiau datele de la periferic
registrul de **ieşire**, în care se scriu datele pentru a le transmite către periferic

- Semnalele de control sunt transmise de UCP pentru a activa perifericul şi pentru a-l informa ce operație trebuie să efectueze
  - Semnalele de control emise sunt specifice pentru fiecare tip de periferic.
- Semnalele de stare sunt utilizate pentru a testa diferite condiții ale interfeței şi ale perifericului
  - De exemplu, calculatorul poate testa dacă un periferic este gata pentru un viitor transfer de date
  - În timpul transferului se pot produce erori care sunt detectate de interfață şi marcate prin setarea unor biți dintr-un registru de stare al interfeței, registru ce poate fi citit de procesor.
- Comenzile pentru date de ieşire fac ca interfața să răspundă prin transferarea datelor de la magistrala de date către registrele sale interne
- Comenzile pentru date de intrare sunt similare cu cele pentru ieşirea datelor, diferind doar sensul de circulație al informației
  - Procesorul testează starea interfeței pentru a verifica dacă datele cerute pot fi transferate către magistrala de date.

Resursele interfetei, prezentate in schema bloc sunt:

-echipamentul de intrare / iesire (poate fi numai de intrare, sau numai de iesire, sau de intrare / iesire);

-UCI reprezinta unitatea de comanda a interfetei;

-RC este registrul de comanda, in care programatorul inscrie printr-o instructiune **out** un cuvant de comanda care determina operatiile realizate la nivelul interfetei. Incarcarea cuvantului de comanda se face prin activarea de catre UCI a unui semnal *out comanda*, la decodificarea adresei corespunzatoare portului de iesire asociat cu acest registru (adresa de port este preluata de pe magistrala de adrese de catre UCI si decodificata, pentru recunoasterea adreselor porturilor din cadrul interfetei);

-RS este registrul de stare, in care UCI inscrie informatiile de stare primite de la echipamentul periferic, dar si informatiile de stare de la nivelul interfetei. Fiecare bit din registru are o semnificatie precisa. Programatorul poate sa citeasca continutul acestui registru prin executia unei instructiuni **in**, cu adresa corespunzatoare portului de intrare asociat registrului. Citirea cuvantului de stare se face prin activarea semnalului de comanda *in stare* de catre UCI, cand continutul registrului este plasat prin intermediul circuitelor tampon cu trei stari pe magistrala de date si este transferat apoi la procesor.

-RDE este registrul de date de iesire in care programatorul inscrie un cuvant de date printr-o instructiune **out** (UCI activand semnalul corespunzator *out data*). Data va fi transferata apoi la echipamentul periferic si scoasa pe suport extern.

-RDI este registrul de date de iesire in care un cuvant trimis de echipamentul periferic este incarcat si apoi citit prin program cu o instructiune **in** (la activarea semnalului *in data*).

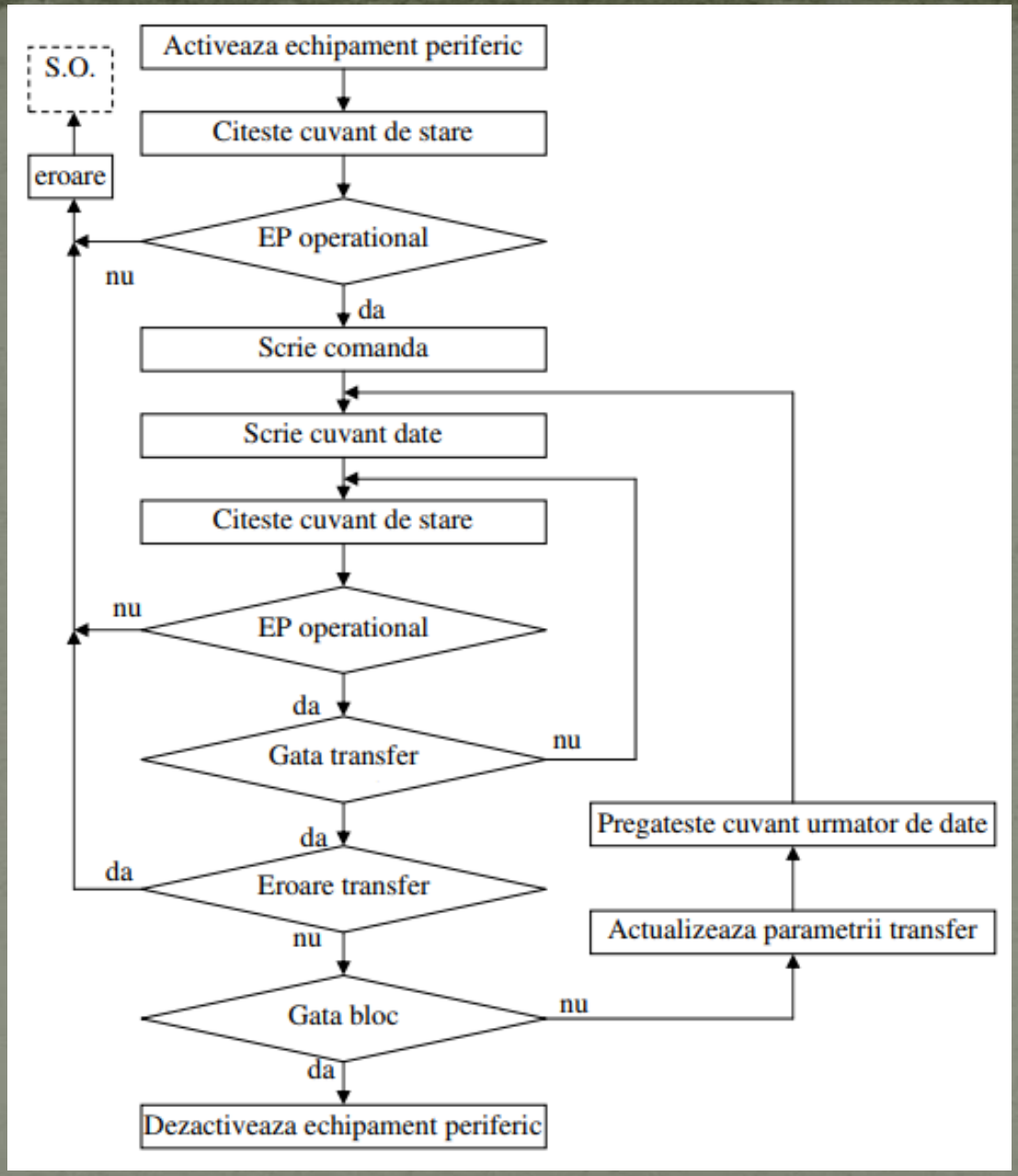
In schema bloc de mai sus este reprezentat si semnalul de intrerupere, care poate fi activat de catre UCI si care este transmis la sistemul de intreruperi si apoi la procesor, in cazul in care transferul de date cu echipamentul periferic se face in intreruperi.

| | |
|---|---|
| MOV DX, adresa port | se încarcă în DX adresa portului de I/O |
| IN AL, DX | se citesc date de la portul adresat |
| MOV [BX], AL | se salvează datele în memorie, la adresa specificată de BX |
| MOV AL, date | se încarcă date în acumulator |
| OUT DX,AL | datele se trimit la portul cu adresa specificată în DX |

Secvența anterioară citeşte date de la un port cu adresa specificată în registrul DX, le salvează în memorie şi scrie apoi date în acelaşi port.

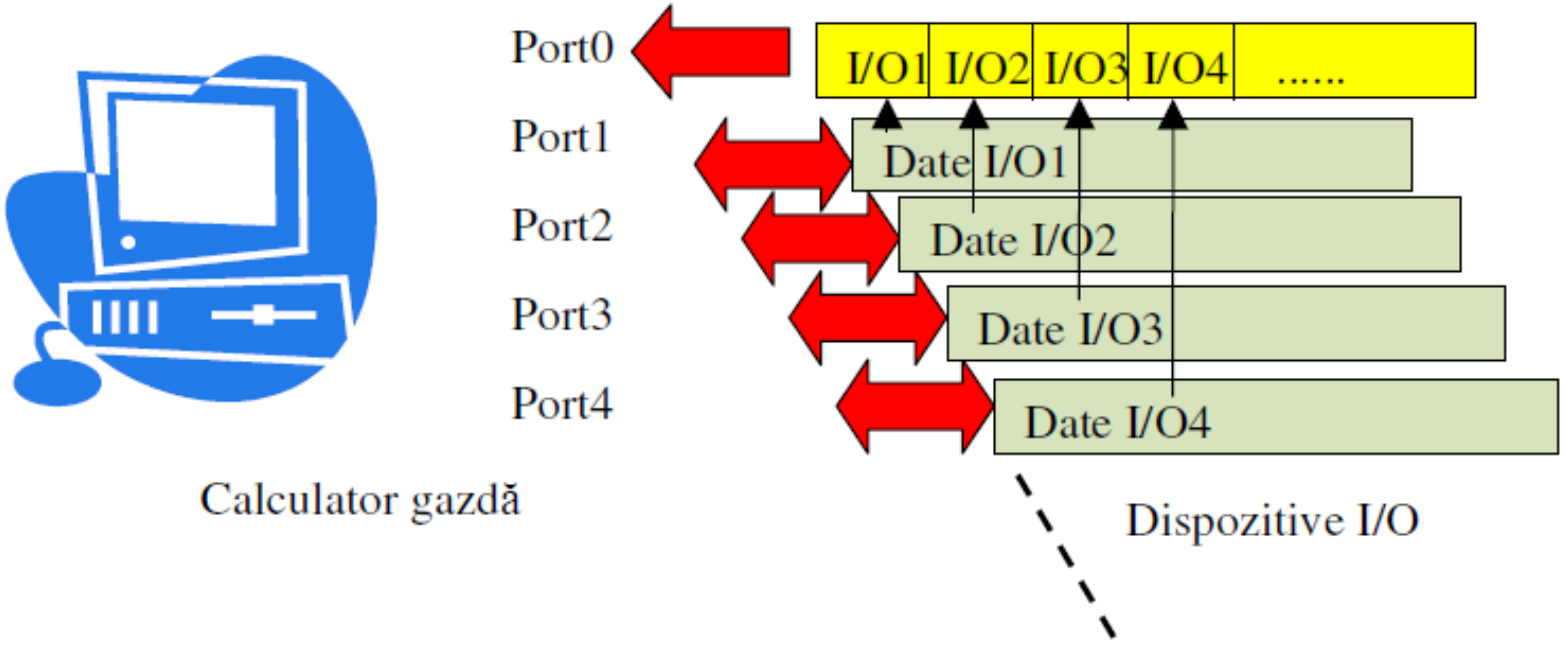
a) *Transfer programat de I/E cu citirea ciclica a starii.* Toate fazele transferului sunt controlate de catre procesor, prin executia unei secvente de program. Cea mai mare parte din timp procesorul asteapta intr-o bucla de program incheierea unui transfer elementar (transferul unui cuvant de date), datorita diferentei mari de viteza intre componentele mecanice ale echipamentului periferic si componentele integral electronice ale unitatii centrale de prelucrare. Pentru a urmari cum se desfasoara transferul de date in acest mod, se va considera scrierea unui bloc de cuvinte la un echipament periferic de iesire.

- Pooling




*Transfer programat de I/E cu citirea ciclica a starii.*

Fiecărui dispozitiv de I/O i se atribuie un bit (fanion) care indică faptul că acesta este gata pentru transfer. Presupunem 8 dispozitive de I/O care au atașate 8 porturi de date și un port pentru citirea fanioanelor, structura fiind dată în figura 2.2.




Transferul prin interogare

Fanioanele sunt interogate ciclic şi se serveşte acel periferic care solicită un transfer de date prin schimbarea stării fanionului. Dacă de exemplu cererea se face cu nivel logic 1 al fanionului, un exemplu de program este:

```
Start:  MOV DX, adresa Port0
        IN AL,DX            se citesc fanioanele
        JN adresa1          la adresa1 este secvența de program de transfer cu I/O1
        RCL AL,01           deplasare AL la stânga
        JN adesa2           la adresa2 este secvența de program de transfer cu I/O2
        ……
        JMP start
```

Se citeşte portul care conține fanioanele, apoi se verifică primul bit. Dacă cel mai semnificativ bit este 1 numărul binar din acumulator este negativ şi dispozitivul solicită un transfer. După testarea tuturor celor 8 fanioane bucla de citire şi testare este reluată. Secvențele de program de transfer pentru fiecare dispozitiv sunt secvențe de transfer programat direct, aşa cum au fost descrise anterior. Desigur că acest program este unul principial, care trebuie completat şi extins pentru a deveni un program funcțional.

**Avantajele** transferului programat sunt simplitatea hardware şi software şi faptul că există controlul strict al timpului în care se face un transfer de date.

**Dezavantajele** transferului programat sunt:

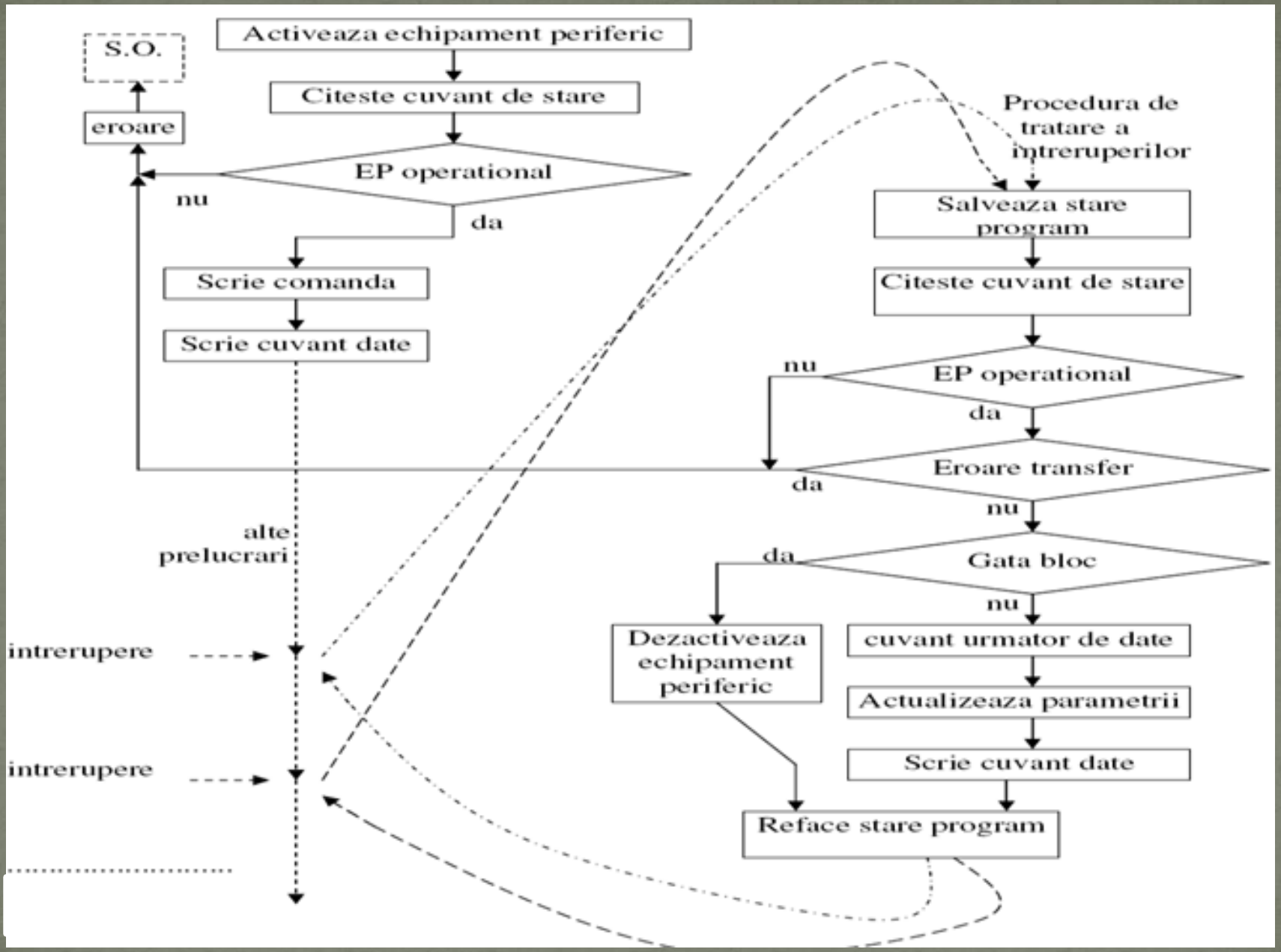
1. Sistemul se ocupă în cea mai mare parte a timpului cu dispozitivele de I/O, devenind dedicat acestei sarcini;
2. Timpul de răspuns la o solicitare este mare, programul de interogare consumând timp.

Ce exemplu cunoasteți în care timpul de execuție al unei secvențe de program este foarte important?

La achiziția semnalelor, la sistemele în timp real eşantionarea trebuie să fie realizată periodic, rutina având aceeaşi durată, altfel conversia va fi eronată.

b) *Transfer programat de intrare / iesire in intreruperi*. Este o solutie mai performanta decat cea precedenta, deoarece procesorul comanda un transfer elementar pentru un cuvant de date, iar in continuare procesorul nu mai asteapta incheierea transferului, realizand prelucrari utile. La incheierea transferului interfata echipamentului periferic genereaza o intrerupere, catre sistemul de intreruperi, care la randul sau genereaza o intrerupere catre procesor. Procesorul isi suspenda prelucrarile in curs si trece la executia unei secvente speciale de program, procedura de tratare a intreruperii. In aceasta procedura procesorul executa o serie de operatii legate de transfer si poate comanda un nou transfer elementar, reluandu-si apoi prelucrarile suspendate.

Pentru exemplificare se considera organigrama operatiilor efectuate de catre procesor pentru scrierea unui bloc de cuvinte la un echipament periferic de iesire in intreruperi


S.O.
Activeaza echipament periferic
Citeste cuvant de stare
eroare
EP operational
nu
da
Scrie comanda
Scrie cuvant date
alte prelucrari
intrerupere
intrerupere
Procedura de tratare a intreruperilor
Salveaza stare program
Citeste cuvant de stare
nu
EP operational
da
Eroare transfer
da
nu
da
Gata bloc
nu
Dezactiveaza echipament periferic
cuvant urmator de date
Actualizeaza parametrii
Scrie cuvant date
Reface stare program

# I/E mapate in memorie

- **In modul de lucru cu intrari/iesiri mapate in memorie** utilizeaza un mecanism cu instructiuni LOAD si SAVE.
- Procesorul va impartii memoria cu alte dispozitive ale SC. Anumite locatii de memorie vor fi considerate ca porturi virtuale de I/E.
- SAVE - scrierea informatiilor din perifericul caruia i s-a alocat acea adresa de memorie.
- Problema apare la viteze diferite de lucru intre procesor si periferic.(Daca perifericul mai lent, atunci trebuiesc generate stari de asteptare pentru procesor)

# I/E prin DMA

- Transferul, prin intermediul UCP, a blocurilor mari de date între memoria principală şi periferice, se face relativ lent, pentru că de fiecare dată informaţia trece şi prin registre ale UCP
- Folosirea DMA conduce la creşterea vitezei sistemului prin degrevarea microprocesorului de controlul transferurilor de date, accesul la memorie fiind efectuat de periferic.
- Când ne referim la DMA, ne referim de fapt la transferuri de date şi la controlul acestora între memorie şi porturile I/E. Activitatea este supravegheată în cazul transferurilor de tip DMA de către un circuit specializat numit **controller DMA**.
- Dacă o setare DMA a fost realizată greşit, de exemplu acelaşi canal DMA a fost alocat pentru mai multe dispozitive, placa nou conectată nu va funcţiona sau chiar va bloca sistemul. În funcţie de aplicaţie, se pot folosi unul sau mai multe canale DMA.

# I/E prin DMA

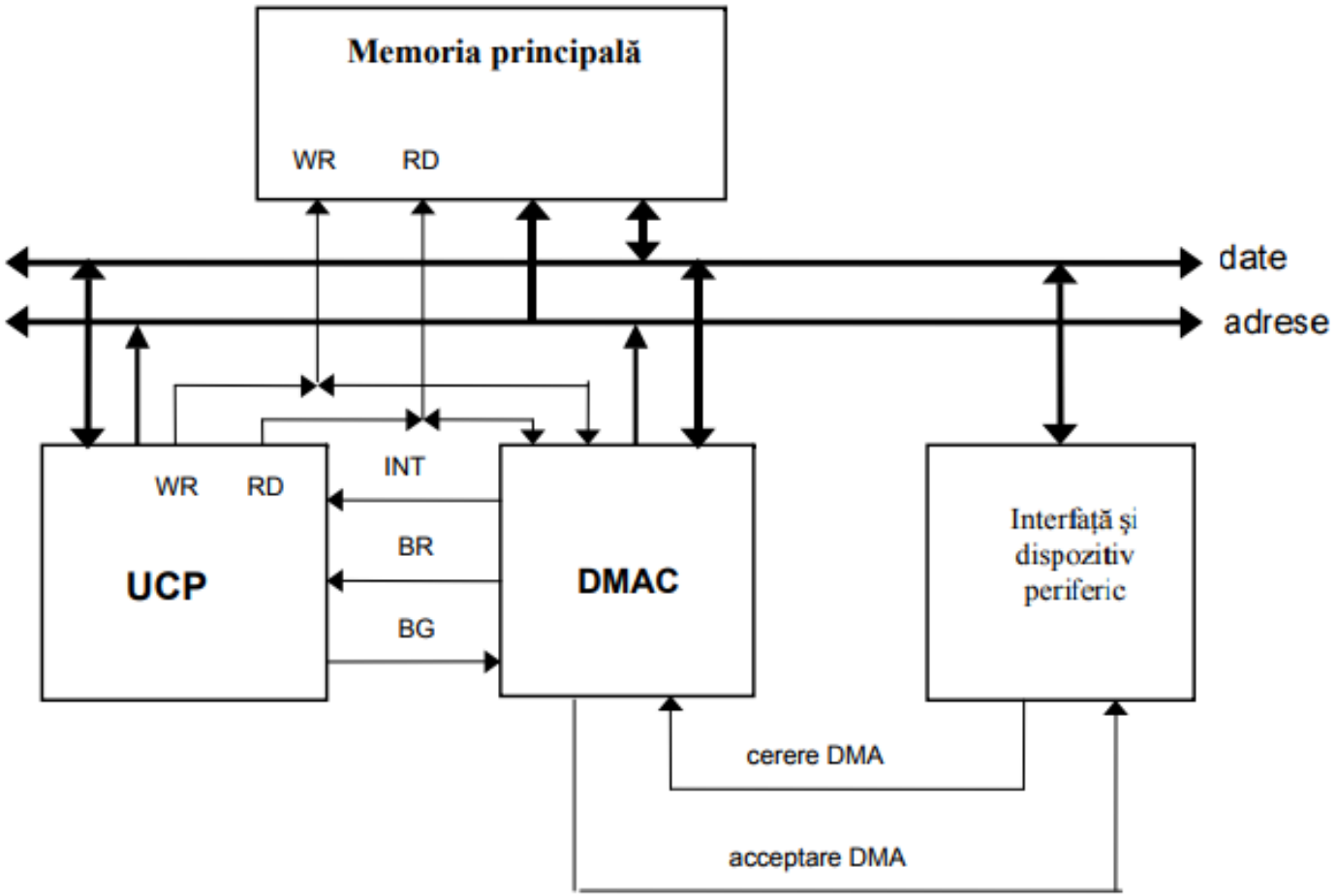
- Accesul direct la memorie este utilizat in cazul transferurilor foarte rapide de date, care se efectueaza de la/la periferice cu viteza de lucru superioare celei a microprocesorului.

- Exista trei metode pentru transferuri DMA:

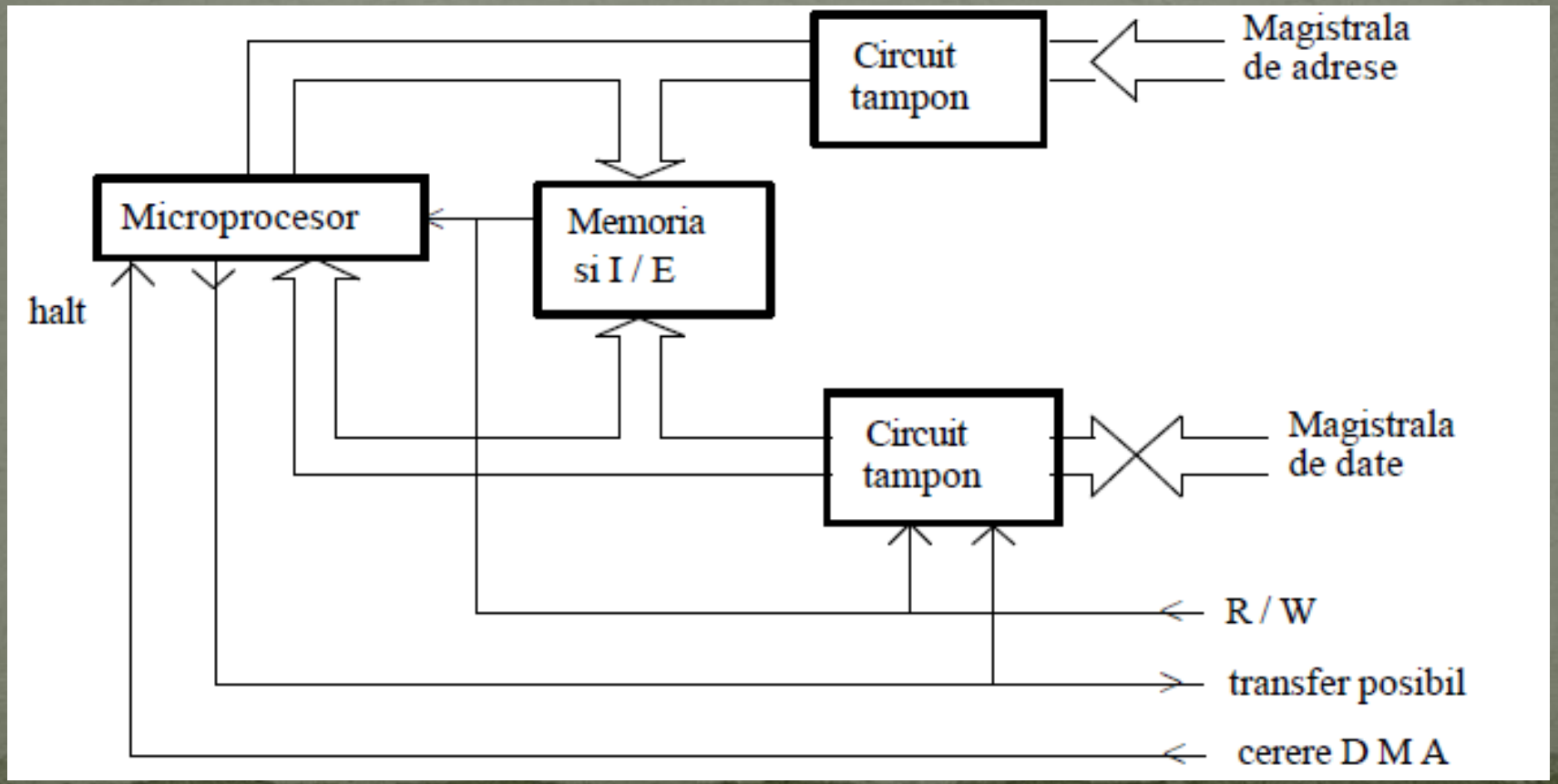
  - **Metoda de oprire a activitatii procesorului (HALT)**
  - **Metoda de oprire a microprocesorului in cadrul instructiunii curente**
  - **Metoda prin care operatiile microprocesorului si ale DMA sunt multiplexate**

- Se presupune în schema bloc următoare că inițierea transferului se face de către periferic (prin intermediul interfeței) care efectuează o cerere de transfer prin acces DMA spre circuitul controler de DMA (DMAC)
- Acesta solicită de la UCP controlul magistralelor prin semnalul de cerere de control a magistralelor, BR (Bus Request)
- La sfârşitul ciclului maşină curent UCP cedează controlul magistralelor, îşi trece ieşirile către acestea în HiZ (stare de înaltă impedanță) şi informează despre aceasta prin semnalul acordare a controlului magistralelor, BG (Bus Grant)
- Circuitul controler DMAC furnizează adresele pe magistrala de adrese, preia controlul semnalelor de scriere (WR) şi citire (RD) şi trimite către periferic semnalul de acceptare a transferului prin DMA
- Când dispozitivul I/O primeşte acest semnal de acceptare, el pune un cuvânt pe magistrala de date (pentru scriere în memoria principală) sau citeşte un cuvânt de pe magistrala de date (pentru citire din memoria principală).

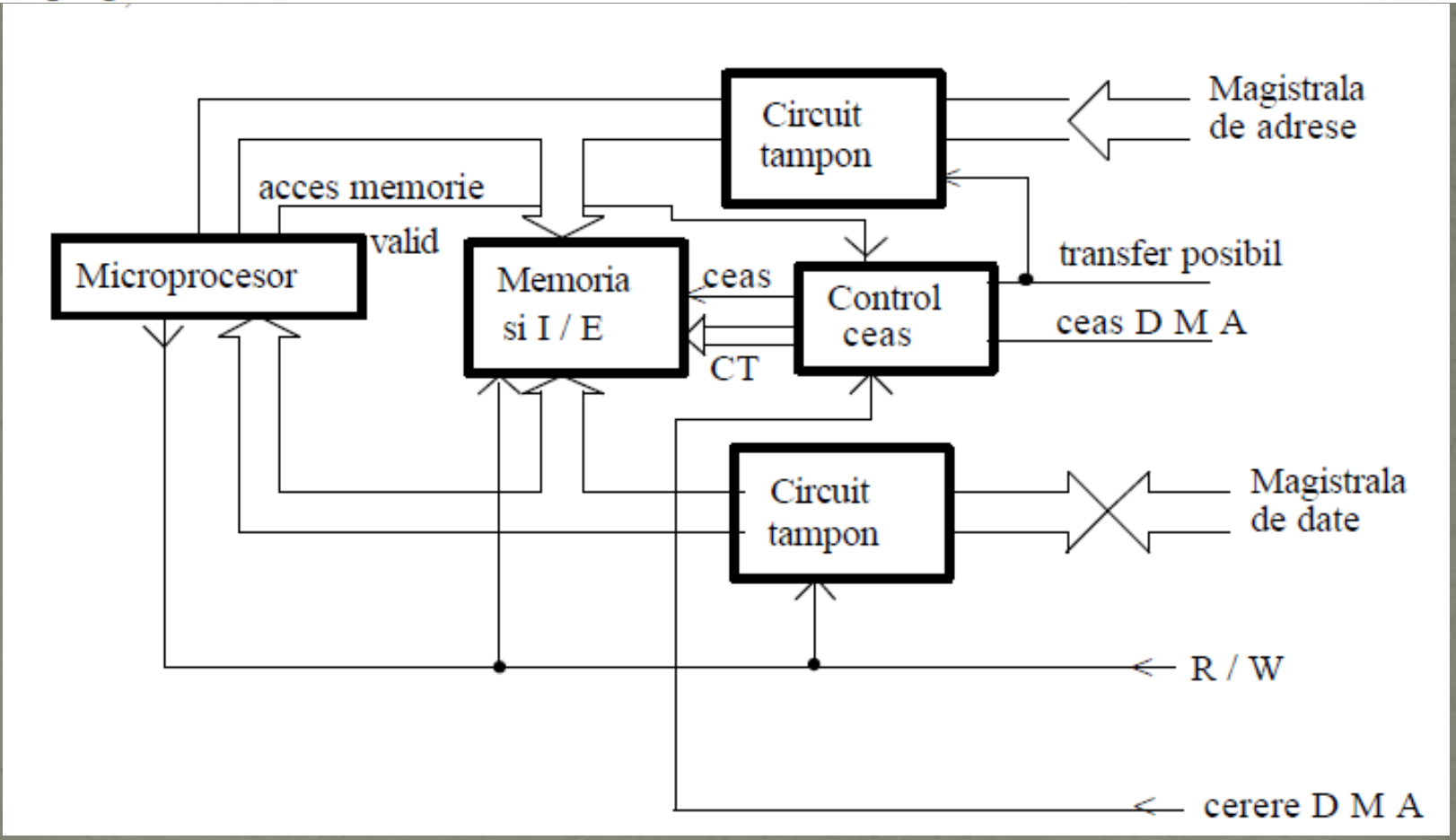
# Exemplu de transfer DMA



***Metoda de oprire a activitatii procesorului (HALT)*** implica, la aparitia unei cereri de acces *DMA*, *terminarea efectuarii instructiunii curente* a procesorului, urmata de *trecerea tuturor pinilor acestuia* (pe magistralele de date, adrese si control) *in starea de mare impedanta* si permite *efectuarea transferurilor DMA* pe aceste magistrale. *Dezavantajul* metodei consta in faptul ca vor fi necesare cateva cicluri-masina din momentul aparitiei cererii de transfer, pana la posibilitatea satisfacerii acesteia. Uneori, aceasta intarziere poate fi prea mare. *Avantajos* este faptul ca, odata obtinut, controlul magistralelor poate fi mentinut pentru transferuri oricat de lungi.



O alta metoda consta in ***oprirea microprocesorului in cadrul instructiunii curente***, *la terminarea ciclului de instructiune curent*. Se obtin timpi de raspuns mult mai buni, deci performante sporite. Se pot transfera deasemenea seturi mari de date, la o astfel de intrerupere a executiei programului.




Metoda lungeste practic durata executarii unor instructiuni in timpul carora procesorul este oprit si se efectueaza transferul de informatii prin *DMA)*.

A treia metoda este complet "*transparenta*" pentru procesor. Acesta nu este oprit, si nici incetinit la executarea programului. Este metoda cea mai rapida, dar si cea mai critica in acelasi timp. Necesita o deosebita atentie la proiectare si memorii cu timpi de acces foarte mici. ***Operatiile microprocesorului si ale DMA sunt multiplexate***, luand in considerare faptul ca transferurile unitatii centrale apar doar pentru anumite faze ale ceasului sistemului.

Daca mai multe dispozitive doresc sa obtina in acelasi timp accesul la un bus de date se obtine “contentious de bus” (controversat).

Daca un dispozitiv are controlul busului iar altul primeste accesul in acelasi timp rezulta “coliziune.”

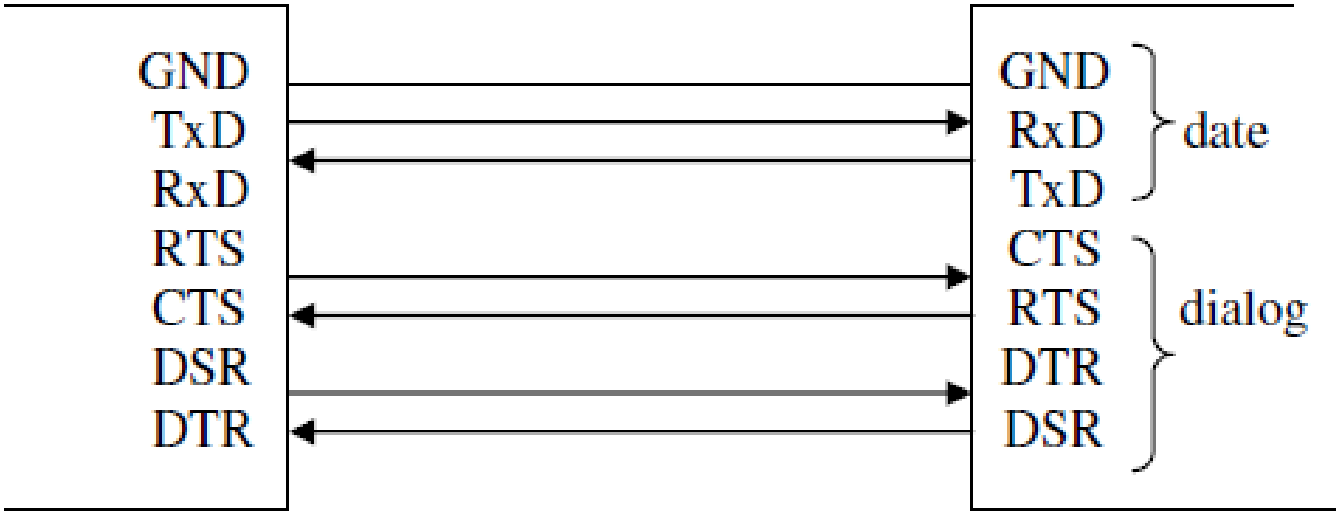
Datorita timpului de acces redus la memorii si a timpului mai scurt de lucru a procesorului rezulta utilizarea in aplicatii TR.

O parte  a memoriei SC este definita ca memorie cu acces DMA.

# Interfata seriala

Se considera un exemplu de interfata in transfer programat si anume interfata seriala. Aceasta permite realizarea de transferuri de date intre sistemul de calcul si un dispozitiv numeric in modul serial ( bit cu bit, transferandu-se un singur bit la un moment dat). Principalul avantaj il reprezinta simplitatea si costul scazut, dar rata de transfer este mai mica in raport cu transferurile paralele.

Pentru comunicatia seriala (standardul RS 232-C) intre doua dispozitive numerice se utilizeaza urmatoarele semnale




*Schimbul de semnale in comunicatia seriala.*

GND (ground) = masa electrica;

TxD (Transmitter Data) = transmisie de date;

RxD (Receiver Data) = receptie de date;

RTS (Request to Send) = cerere pentru emisie;

CTS (Clear to Send) = anulare in vederea transmisiei;

DSR (Data Set Ready) = semnifica conectare la linie (echipament operational);

DTR (Data Terminal Ready) = dispozitivul cu care se face transferul de date este conectat la linie.

Datele transmise pe linia seriala sunt organizate conform diagramei de semnale urmatoare

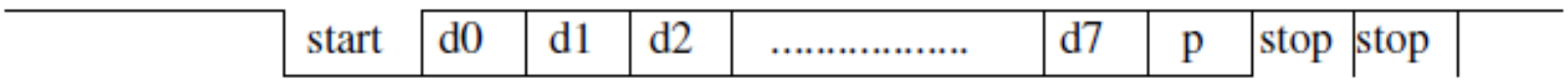



*Diagrama de semnal pentru transmisia unui cuvant pe linia seriala.*

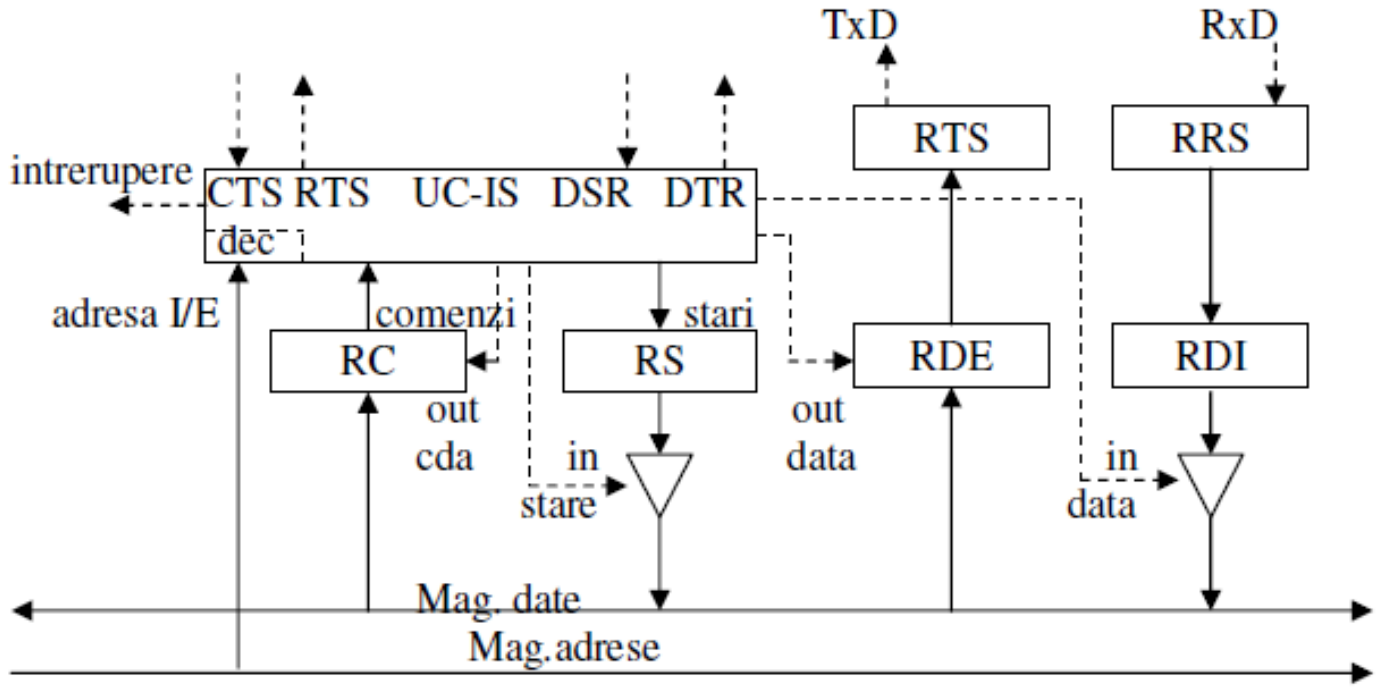
-start = bitul care marcheaza inceputul trransmisiei unui cuvant (octet);

-d0-d7 = bitii de date;

-p = bitul de paritate, calculat dupa paritatea para sau impara;

-stop = unul sau doi biti de stop, care marcheaza sfarsitul cuvantului de date transmis.

Schema bloc a interfetei seriale este prezentata in figura urmatoare




*Schema bloc a interfetei seriale.*

Resursele prezentate in schema bloc sunt:

RRS = registrul de receptie seriala, este un registru cu deplasare, avand intrarea seriala conectata la linia seriala de receptie, RxD. Realizeaza impachetarea bitilor receptionati in cadrul unui cuvant care va fi transferat apoi in paralel.

RDI = registrul date de intrare, primeste in paralel cuvantul de date receptionat de pe linia seriala. Continutul sau poate fi citit in paralel prin executia unei instructiuni **in**, cu adresa de port corespunzatoare registrului.

RDE = registrul date de iesire, este inscris prin executia unei instructiuni **out** cu un cuvant care va fi apoi transmis pe linia seriala.

RTS = registrul de transmisie seriala, primeste cuvantul de date de transmis de la registrul RDE. Fiind un registru cu deplasare, informatia este furnizata bit cu bit pe linia seriala de transmisie TxD.

UC-IS = unitatea de comanda a interfetei seriale.

RC = registrul de comanda (la fel ca la interfata generala prezentata anterior). Anumiti biti au o importanta deosebita pentru functionarea interfetei seriale. Astfel,

| ..... | DTR | L1 | L0 | STOP | PAR | ..... |
|---|---|---|---|---|---|---|

DTR : comanda linia de dialog cu acelasi nume, pentru conectarea la linie a dispozitivului;

L1,L0 : determina lungimea cuvantului transmis pe linia seriala;

STOP : specifica numarul de biti de stop utilizati in transmisie si care se verifica la receptie;

PAR : determina paritatea utilizata pentru verificarea datelor transmise (paritate para sau impara).

RS = registrul de stare (la fel ca la interfata generala). Anumiti biti din acest registru sunt importanti pentru determinarea starii interfetei:

| ..... | DSR | TxRDY | TxEMPTY | RxRDY | PE | FE | OE | ..... |
|---|---|---|---|---|---|---|---|---|

DSR : furnizeaza starea liniei de dialog cu acelasi nume, avand semnificatia de echipament operational;

TxRDY (Transmitter Ready) : specifica daca registrul RDE este liber pentru ca procesorul sa trimita un nou cuvant de date care sa fie apoi transmis pe linia seriala;

TxEMPTY (Transmitter Empty) : specifica daca ambele registre folosite la transmisie RDE si RTS, sunt libere, deci interfata nu are nimic de transmis la momentul respectiv;

RxRDY (Receiver Ready) : specifica daca s-a receptionat un cuvant de date de pe linia seriala si acesta se poate citi din registrul RDI printr-o instructiune **out**;

PE (Parity Error) : activarea acestui bit de stare indica faptul ca la receptia ultimului cuvant de date a aparut o eroare de paritate;

FE (Framming Error) : indica o eroare datorata diferentei mari de frecventa intre ceasul de la transmisie si ceasul de la receptie, cand in locul unui bit de stop se detecteaza 0 logic (posibil bitul de paritate de la cuvantul curent sau bitul de start de la cuvantul urmator);

OE (Overrun Error) : eroare aparuta din programarea interfetei seriale. Cuvantul de date receptionat anterior si aflat in RDI nu a fost citit in timp util de catre procesor (printr-o instructiune **in**), s-a receptionat de pe linia seriala un cuvant nou, care a fost inscris in RDI peste cuvantul precedent  Astfel s-a pierdut un cuvant din cadrul blocului transferat.

Vehicularea datelor se realizeaza prin modul de lucru cu intrerupere.

**Intreruperea** este un semnal electronic generat de catre o unitate functionala, de exp. canal sau controler de periferice si acest semnal este transmis spre CPU pentru a provoca o ruptura de secventa in vederea executie unui program prioritar care trateaza cauza intreruperii.

Semnale de intreruperi:

- interne: depasire de capacitate, coduri de operatii inexistente, erori de adresare, pana de curent.
- externe: starea unei unitati periferice, sfarsitul unui transfer de date.

Tratarea intreruperii:

- oprirea executiei programului in curs
- salvarea starii sistemului
- executarea programului de tratare a intreruperii
- restaurarea starii sistemului
- reluarea executiei programului intrerupt.

- Cauzele intreruperilor sunt afisate in **vector indicatori**, iar programul care trateaza intreruperea trebuie sa testeze acesti indicatori.

Majoritatea sistemelor de calcul moderne sunt prevăzute cu sisteme de întrerupere ierarhizate [priority interrupt systems], acestea fiind sistemele cu nivele de prioritate.

Problemele care trebuiesc rezolvate sunt următoarele:

- sosirea mai multor semnale de întrerupere în timpul execuţiei unei instrucţiuni;
- sosirea unui semnal de întrerupere în timpul execuţiei unui program de prelucrare a unei întreruperi anterioare.

În aceste sisteme, fiecare nivel este asociat unui anumit număr de întreruperi, fiecărui nivel îi corespunde un anumit nivel de prioritate, iar orice program poate fi întrerupt în vederea realizării unei întreruperi mai prioritare.

Un sistem de întrerupere modern trebuie să permită programatorului următoarele acțiuni:

- invalidarea/activarea [disable/enable] sistemului de întrerupere;
- mascarea/demascarea individuală a întreruperilor;
- stabilirea unei ierarhii în mulţimea cauzelor intreruperilor și definirea mai multor nivele de prioritate, de preferinţă dinamic;
- asocierea unui program specific fiecărei întreruperi, permiţând acțiuni elementare realizabile într-o singură instrucţiune;
- posibilitatea de utilizare a tuturor registrelor care caracterizează starea mașinii și refacerea acestora la sfârşitul programului de întrerupere.

# Metoda întreruperilor externe

- Prin aceasta metoda dispozitivul periferic atentioneaza aplicatia generând întreruperi externe. Tratarea prompta a întreruperilor de la toate dispozitivele este posibila atât timp cât cererile catre procesor sunt rezonabile, acesta este capabil sa lanseze rapid rutinele de tratare iar timpul de executie al acestora este suficient de mic.

- În particular, metoda întreruperilor externe este de preferat în aplicatii care cer precizie pentru timpul de achizitie de date si control, în timp ce procesorul executa si alte sarcini. De asemenea, este utila daca mai multe dispozitive **solicita asincron servicii**, la intervale de timp nepredictibile.

# Sistemul de intreruperi

- Pentru programarea întreruperilor sunt necesare:

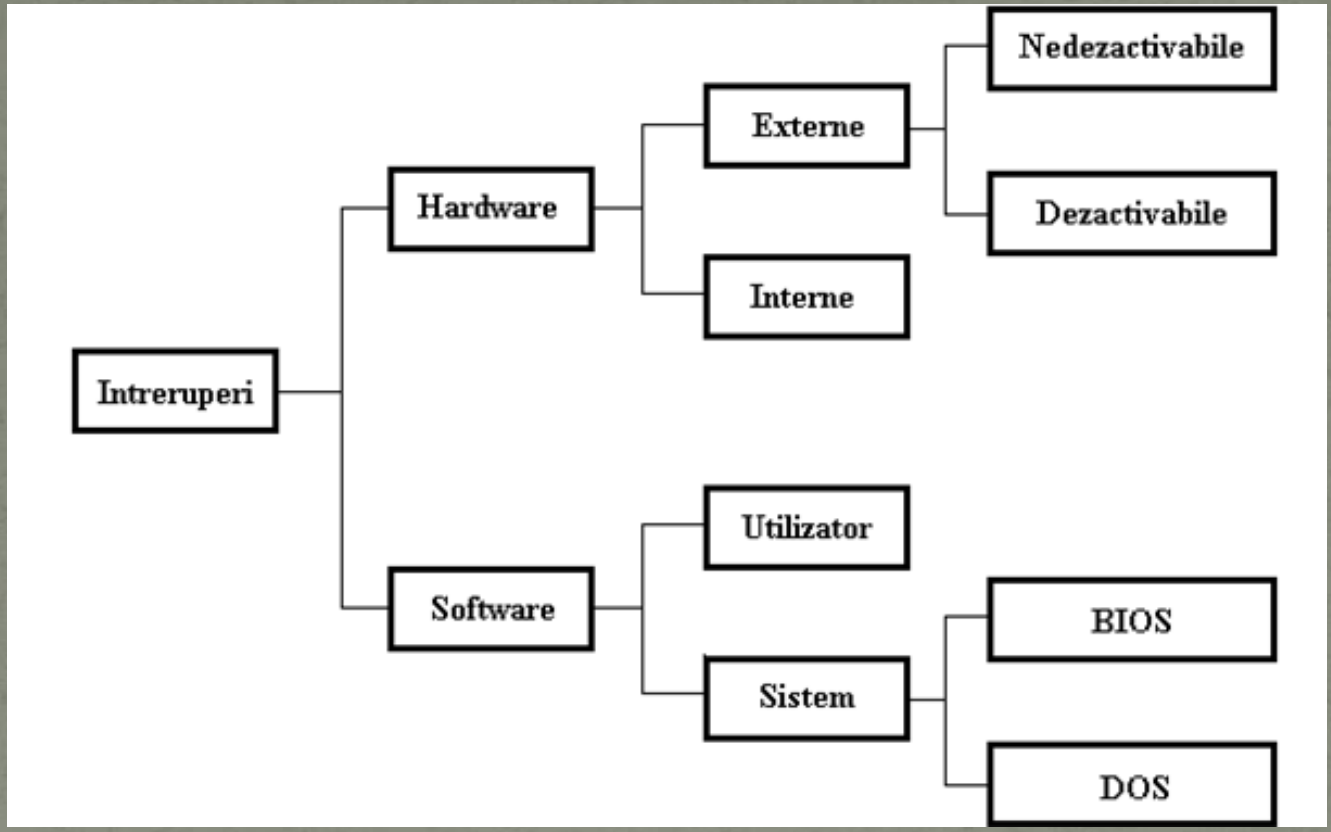
  - cunoasterea sistemului de întreruperi al calculatorului si modul de programare al acestuia.

  - cunoasterea limbajului de asamblare al procesorului sau cel putin cuvintele cheie si procedurile speciale pentru întreruperi furnizate de anumite medii de dezvoltare pentru limbaje evoluate.

  Unele limbaje precum C/C++, PASCAL, ADA furnizeaza mijloace de tratare a întreruperilor cu proceduri care realizeaza citirea/scrierea prin **adresarea absoluta** a locatiilor de memorie, operatii de I/E la nivel fizic prin **programarea porturilor sau locatiilor de memorie**, secvente de I/E din procedurile de tratare a întreruperilor.

- Succesiunea executiei instructiunilor de catre procesor depinde de aparitia evenimentelor care declanseaza proceduri de tratare a întreruperilor. Greselile în tratarea sistemului de întreruperi al calculatorului provoaca erori grave în executia programului si afecteaza negativ functionarea sistemului, ceea ce face ca programele care utilizeaza întreruperile sa fie greu de depanat.

- Pot folosi întreruperi:
  - Hardware şi software
    - cele software implică un ceas
  - Periodice
    - numite sisteme cu timp de execuție constant
  - Aperiodice
    - numite sisteme sporadice
  - Mixte
    - numite sisteme hibride


Intreruperi
Hardware
Externe
Nedezactivabile
Dezactivabile
Interne
Software
Utilizator
Sistem
BIOS
DOS

- Diferența între întreruperile **hardware** și **software** este dată de entitatea care generează întreruperea. Dacă este provocată de un dispozitiv hardware, atunci este o întrerupere hardware, iar dacă este provocată de un program, este o întrerupere software.

- Astfel, întreruperile **software** apar ca urmare a execuției unor instrucțiuni, cum ar fi INT, DIV, IDIV.

- Întreruperile **hardware** sunt interceptate de rutine speciale care se execută în mod continuu cu scopul de a le detecta și de a efectua acțiunile corespunzătoare. Ele sunt de 2 tipuri:

  - Întreruperile hardware externe
  - Întreruperile hardware interne

- **Întreruperile hardware externe** sunt provocate de semnale electrice care se aplică pe intrările de întreruperi INTR şi NMI ale procesorului.

- **Întreruperile hardware interne** apar ca urmare a unor condiții speciale de funcționare a procesorului (cum ar fi execuția pas cu pas a programelor, împărțirea la 0);

- Nivelele predefinite de întrerupere sunt:
- 0 - depăşire la împărţire (cauze posibile; instrucţiunile DIV sau IDIV);
- 1 - execuţie pas cu pas (cauză posibilă: bistabilul TF = 1);
- 2 - întrerupere externă nedezactivabilă (cauză posibilă: semnal electric pe linia de întrerupere nedezactivabilă NMI);
- 3 - execuţie pana la o anumita linie (cauză posibilă: instrucţiunea INT 3);
- 4 - depăşire (cauză posibilă: instrucţiunea INTO).

IDIV - (**Integer DIVide)**, daca destinatia specificata in instructiune pentru a memora catul impartirii nu are dimensiunea suficienta pentru acest scop.

INTO - (**INTerrupt on Overflow)**

**Trap Flag**. Daca TF=1, procesorul intra in modul de operare pas cu pas in care CPU genereaza automat o intrerupere interna dupa fiecare instructiune pentru a permite examinarea starii programului si deci depanarea acestuia.

Cererea de intrerupere nemascabila (NMI) este utilizata, de obicei, pentru a semnala microprocesorului aparitia unui eveniment "catastrofal" ce semnifica existenta unui pericol major pentru buna functionare a sistemului.

Exemple tipice sunt cadere a tensiunii de alimentare, aparitia unei erori de memorie.

Intreruperi hardware de la ceasul de timp real şi de la tastatura.

Întreruperile software în gama 20H - 2FH sunt folosite de sistemul de operare DOS, iar cele în gama 10H - 1AH de către subsistemul de intrări - ieşiri BIOS.

- **Întreruperi software**
- **Instructiunea INT**
- INT nr_intrerupere (un numar cuprins intre 0 si 255);
- Intreruperile opresc programul din rulare, executa un anumit cod si apoi permit continuarea rularii programului. Cu ajutorul apelului intreruperilor aplicatia poate sa comunice cu sistemul de operare, si cu unele componente hardware ale calculatorului (mouse-ul, tastatura, imprimanta, etc). Fiecare intrerupere are asociat un numar.

*Exemple*:

0x9 : intreruperea pentru tastatura;

0x10 : intreruperea care ofera serviciile BIOS pentru accesul la placa video;

0x21 : intreruperea sistemului de operare DOS;

0x33 : intreruperea pentru mouse;

- In DOS pentru initializarea mouse-ului se va apela functia numarul 0 a intreruperii 0x33 :

```
asm
{
    mov ax, 0
    // seteaza registrul ax cu valoarea 0 pentru ca
    // driverul ce gestioneaza intreruperea 0x33
    //sa stie ca trebuie sa initializeze mouse-ul
    int 0x33
}
```

Pentru afisarea mouse-ului :

```
asm
{
    mov ax, 1
    int 0x33
}
```

Pentru ascunderea cursorului de la mouse :

```
asm
{
    mov ax, 2
    int 0x33;
}
```

- Un exemplu este INT 21h care este o întrerupere a serviciului DOS. Când rutina este apelată, aceasta va citi valoarea stocată în ah, şi va lansa rutina corespunzatoare.
- În exemplul următor este invocată întreruperea 21h.

```
   :
mov   ax, 4c00h
int   21h
   :
```

- Aceste 2 linii cer sistemului de operare să încheie execuţia programului.
- Numărul întreruperii nu este suficient. Întreruperile se comportă diferit în funcţie de ce număr de serviciu este apelat. În acest caz, numărul serviciului este plasat în ah şi este egal cu 4ch. (Observaţie: AX=4c00h înseamnă AH=4ch şi AL=00h).

**Aplicaţii:**

Blocarea programului până când este apăsat un buton al mouse-ului (aplic1.asm).

```
start:
mov AX,5        ;5 in AX spune intreruperii 33
                ;sa preia informatiile despre butoanele
                mouse-ului
int 33h         ;apel intrerupere
cmp AX,1b       ;in ax este preluata starea butonului de
                mouse
jl  start       ;daca nu este nici un buton apasat se
                intoarce la eticheta start
                ;AX va fi 0 cat timp nu se apasa nici un
                buton
mov ah,004Ch    ;terminare program
int 21h
end
```

- INT 33 - Mouse Function Calls
- For more information see the following topics:
- INT 33,0 Mouse Reset/Get Mouse Installed Flag
- INT 33,1 Show Mouse Cursor
- INT 33,2 Hide Mouse Cursor
- INT 33,3 Get Mouse Position and Button Status
- INT 33,4 Set Mouse Cursor Position
- INT 33,5 Get Mouse Button Press Information
- INT 33,6 Get Mouse Button Release Information
- INT 33,7 Set Mouse Horizontal Min/Max Position
- INT 33,8 Set Mouse Vertical Min/Max Position
- INT 33,9 Set Mouse Graphics Cursor
- INT 33,A Set Mouse Text Cursor
- INT 33,B Read Mouse Motion Counters
- INT 33,C Set Mouse User Defined Subroutine and Input Mask
- INT 33,D Mouse Light Pen Emulation On
- INT 33,E Mouse Light Pen Emulation Off
- INT 33,F Set Mouse Mickey Pixel Ratio
- INT 33,10 Mouse Conditional OFF

• INT 33,13 Set Mouse Double Speed Threshold
• INT 33,14 Swap interrupt subroutines
• INT 33,15 Get mouse driver state and memory requirements
•INT 33,16 Save mouse driver state
•INT 33,17 Restore mouse driver state
•INT 33,18 Set alternate subroutine call mask and address
•INT 33,19 Get user alternate interrupt address
• INT 33,1A Set mouse sensitivity
•INT 33,1B Get mouse sensitivity
•INT 33,1C Set mouse interrupt rate (InPort only)
•INT 33,1D Set mouse CRT page
•INT 33,1E Get mouse CRT page
•INT 33,1F Disable mouse driver
•INT 33,20 Enable mouse driver
•INT 33,21 Reset mouse software
•INT 33,22 Set language for messages
•INT 33,23 Get language number
•INT 33,24 Get driver version, mouse type & IRQ number
- function is specified in AX - a mickey is 1/200 inches

Ordinea de prioritate este:

- **I. intrerupere interna (cu exceptia pas-cu-pas)**
- **II. NMI**
- **III. INTR**
- **IV. intreruperea pas-cu-pas**

# Metoda I/E buffer-ate

- Aceasta metoda îmbina avantajele polling-ului cu avantajele întreruperilor. Este utilizata în aplicatii precum bucle de control sau în aplicatii de supraveghere, unde datele sunt prelucrate si afisate în paralel cu achizitia lor.
- Implementarea uzuala a **intrarilor bufferate** consta în*:*
  - o rutina de tratare a întreruperilor care citeste (achizitioneaza de la un dispozitiv extern) datele si le memoreaza într-un buffer circular;
  - un task scris ca o bucla polling în care aplicatia asteapta datele iar atunci când sunt în memorie le prelucreaza.

- **Contorul de inserare** în buffer este gestionat de **rutina de tratare a întreruperilor** si indica mereu urmatoarea locatie în care trebuie depuse datele achizitionate. Contorul este incrementat circular.

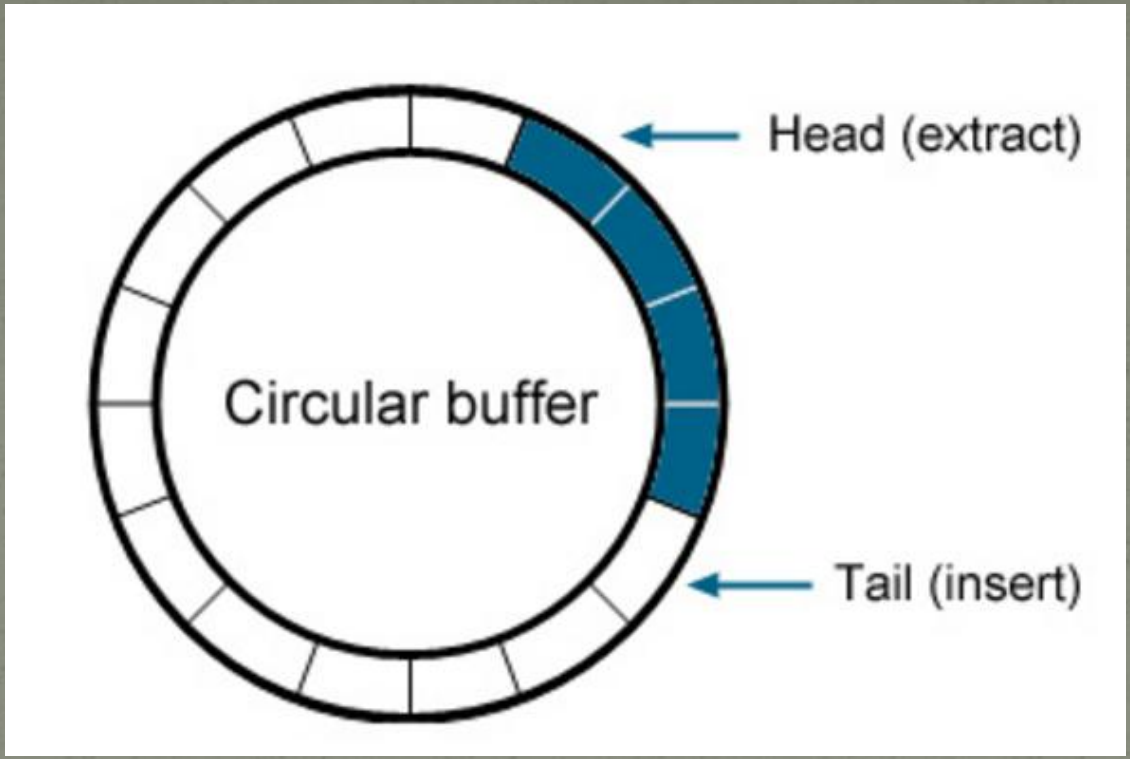
  **Contorul de extragere** este gestionat de **task-ul** care efectueaza în polling prelucrarea datelor din buffer.

  Permanent, task-ul principal compara cele doua contoare: daca acestea sunt diferite, exista date noi între locatiile indicate de contorul de extragere (care indica cea mai veche data depusa în buffer si netratata) si contorul de inserare.

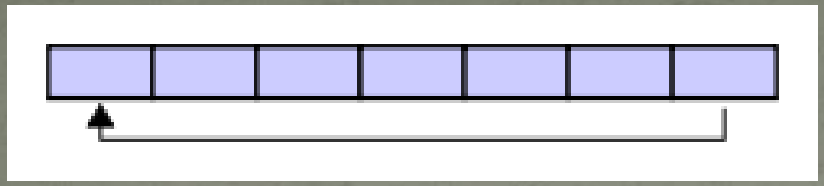
  Daca nu exista date noi procedura polling fie asteapta date, fie efectueaza alte prelucrari. Trebuie sa existe un echilibru între rata de achizitie si viteza de prelucrare astfel încât buffer-ul sa nu contina mai mult decât câteva intrari noi, iar contorul de inserare sa nu depaseasca pe cel de extragere (circular).

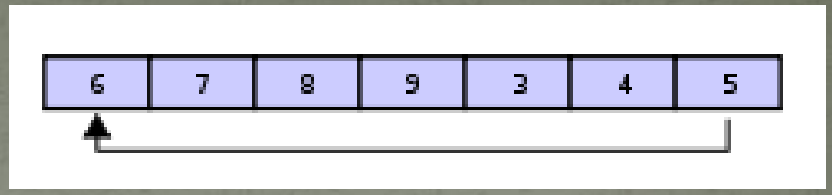
# Metoda I/E buffer-ate

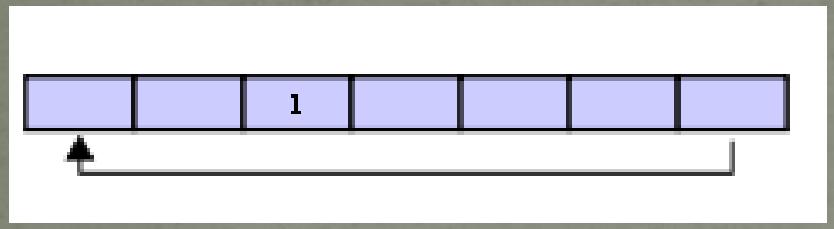
- **Iesirile buffer-ate** pot fi tratate similar. **Task**-ul principal depune datele într-un buffer circular (container), pe masura ce ele sunt prelucrate, iar **rutinele de tratare a întreruperilor** le extrag si le trimit catre dispozitivul de iesire. În acest caz, contorul de inserare este mentinut de task-ul principal, iar cel de extragere de rutina de tratare a întreruperii.
- Daca datele trebuiesc transferate catre dispozitivul extern la intervale de timp egale (de exemplu pentru generarea unor forme de unda), task-ul trebuie sa fie capabil sa depuna date suficient de rapid în buffer, pentru ca acesta sa nu fie vid atunci  când se executa rutina care transfera o noua data catre dispozitiv.



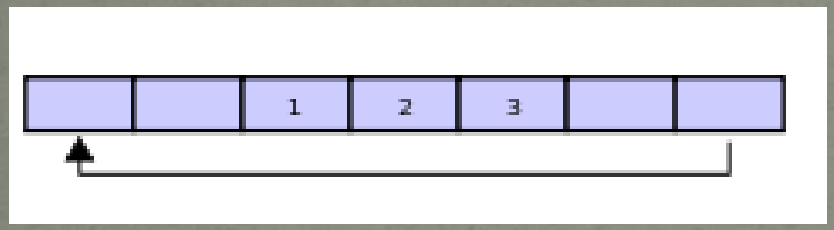

Bufferul circular este umplut cu date exact ca un buffer simplu, dar cand se ajunge la sfarsitul lui si mai exista date de depus, acest lucru se realizeaza de la inceputul bufferului.



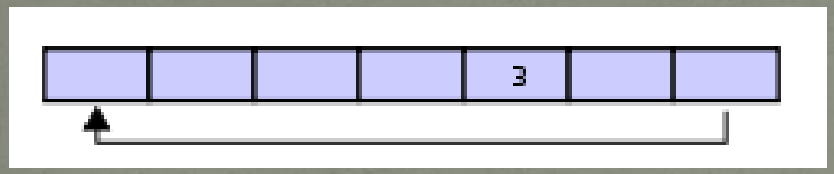

6
7
8
9
3
4
5


1


6
7
8
9
A
B
5


1
2
3


7
8
9
A
B


3


Indicatori intr-un buffer partial plin


Indicatori intr-un buffer plin cu 4 prescrieri

Sintaxa generală pentru declararea unei structuri este următoarea:

```
struct nume_structura {
                        tip1 camp1;
                        tip2 camp2;
                        ...
                        tipN campN;

                      } variabila1, ..., variabilaN;
```

În această declarație, *nume_structura* este opțional dar dacă se specifica, se poate folosi pentru a crea ulterior și alte variabile de acest tip, cu sintaxa:

```
struct nume_structura var1, var2, ..., varN;
```

Lista de variabile *variabila1 ... variabilaN* care să se creeze imediat după declararea structurii este opțională.

Câmpurile unei structuri pot fi de orice tip, inclusiv alte structuri:

Sintaxa completă pentru declararea unei enumerări este următoarea:

```
enum nume_enumerare  {
                        constanta1 = valoare1,
                        constanta2 = valoare2,
                        ...,
                        constantaN = valoareN
                     } variabila1, ...,variabilaN;
```

În aceasta declarație, *nume_enumerare* este opțional, dar dacă se specifică, se pot crea ulterior și alte variabile întregi asociate acestui tip (enum zile azi; enum zile ieri).

De asemenea, valorile *valoare1 ... valoareN*, asociate fiecărei constante sunt opționale. Dacă nu se specifică explicit, se aplică următoarea regulă:

- pentru prima constantă, dacă nu se specifică explicit valoarea, i se atribuie valoarea 0
- pentru celelalte constante, dacă nu au specificată explicit valoarea dorită, li se atribuie valoarea constantei precedente + 1

```c
#define MAX_ITEMS    10
typedef struct circularQueue_s
{
    int     first;
    int     last;
    int     validItems;
    int     data[MAX_ITEMS];
} circularQueue_t;
```

```c
void initializeQueue(circularQueue_t *theQueue);

int isEmpty(circularQueue_t *theQueue);

int putItem(circularQueue_t *theQueue, int theItemValue);

int getItem(circularQueue_t *theQueue, int *theItemValue);

void printQueue(circularQueue_t *theQueue);
```

```c
void initializeQueue(circularQueue_t *theQueue)
{
    int i;
    theQueue->validItems  =  0;
    theQueue->first       =  0;
    theQueue->last        =  0;
    for(i=0; i<MAX_ITEMS; i++)
    {
        theQueue->data[i] = 0;
    }
    return;
}
```

```
enum gen_substantiv   {
                            neutru,          /* valoarea implicită  0
*/
                            feminin = 12,    /* valoarea explicită 12
*/
                            masculin         /* valoarea implicită 13
*/
                        } gen1, gen2;        /*  2 variabile întregi
*/
```

Evident, aceeaşi constantă (simbol) nu poate să apară în mai multe declaraţii de enumerări:

```
enum zile { luni, marti, miercuri, joi, vineri, sambata, duminica };

/* eroare, constantele sambata şi duminica au fost definite deja */
enum zile_weekend { sambata, duminica };
```

```
enum culoare { black, blue, green, cyan, red, magenta, yellow, white
};

struct punct {
                int x;
                int y;
            };

struct punct_colorat {
                      struct punct p;
                      enum culoare c;
                    };

int main ()
{
      struct punct lista_puncte [100]; /* tablou cu elemente
structurate */

      struct punct_colorat pct;
      struct punct_colorat *ptr;

      lista_puncte [0].x = 10;
      lista_puncte [0].y = 11;

      pct.p.x = 21;
      pct.p.y = 10;

      ptr = &pct;      /* ptr indica spre pct */

      ptr->p.x = ptr->p.x + 21;
      ptr->p.y = ptr->p.y + 10;
      return 0;
}
```

- în cazul unei uniuni, toate campurile partajeaza aceeasi zonă de memorie; dimensiunea în octeți a unei uniuni este egală cu dimensiunea în octeți a celui mai mare câmp

Sintaxa pentru declararea unei uniuni este următoarea:

```
union nume_uniune {
                        tip1 camp1;
                        tip2 camp2;
                        ...
                        tipN campN;

                } variabila1, ..., variabilaN;
```

Ca şi în cazul structurilor, numele nume_union este opțional, dar dacă se specifică se poate folosi pentru declararea ulterioară de variabile de acest tip. De asemenea, lista de variabile care să se creeze odată cu declararea uniunii, *variabila1 ... variabilaN* este opțională. Câmpurile se accesează cu operatorul "." sau "->" în cazul accesării indirecte prin intermediul unui pointer.

Dacă într-un program se dorește accesarea valorii octeților care compun un întreg, se poate proceda în felul următor:

```
union întreg {
                int val;
                char octeti [2];
            };

int main ()
{
      union întreg i;

      printf ("Dați valoarea întregului: ");
      scanf ("%d", &i.val);

      printf ("Octetii sunt: low = %d  high = %d \n",
                        i.octeti [0], i.octeti [1]);
      return 0;
}
```

Dacă într-un program se lucrează frecvent cu variabile de tipuri structurate (tablouri, structuri, uniuni) sau variabile de tipuri primitive care au o anumită semnificație pentru programator, se poate îmbunătăți claritatea programelor utilizând declarații de tipuri de date definite de programator. Limbajul C permite declararea unor tipuri de date utilizând directiva *typedef*. În urma declarării unui tip de date, se pot crea ulterior oricâte variabile de tipul respectiv dorim.

```
typedef definiție_tip nume_tip;
...
nume_tip variabila1, variabila2, tab [123];
```

**Exemple :**

```
/* Declarații de tipuri */

typedef int vârsta;

typedef int matrice [10][10];

typedef enum
{
      luni,
      marti,
      miercuri,
      joi,
      vineri,
      sambata,
      duminica
} zile;
```

```
typedef struct
{
      char nume [80];
      char adresa [80];
      int vârsta;
} persoana;

/* Declaraţii de variabile */

vârsta v1, v2;
matrice m1, m2;
zile z1, z2;
persoana lista [100];
```

1. Să se scrie un program în C care să permită efectuarea unor operații asupra unor numere complexe. Programul trebuie să conțină definirea tipului de dată *nr_complex* și următoarele funcții pentru operațiile cu numere complexe:

```
void        citire      (nr_complex *rezultat);
void        afisare     (nr_complex n);
void        adunare     (nr_complex n1, nr_complex n2, nr_complex
*rezultat);
void        scadere     (nr_complex n1, nr_complex n2, nr_complex
*rezultat);
void        inmultire   (nr_complex n1, nr_complex n2, nr_complex
*rezultat);
double      modul       (nr_complex n);
void        conjugat    (nr_complex n, nr_complex *rezultat);
```

Programul trebuie să conțină un meniu cu următoarele opțiuni:

```
1. Suma
2. Diferenta
3. Produsul
4. Modulul
5. Conjugatul
0. Exit
```

- Exemplu: Inregistrare video cu **buffer circular**

  Prin acest tip de data structurata putem sa inregistram un fisier cu o lungime specificata (5 min). Este foarte utila daca de exemplu vrem sa lansam in executie un proces de inregistrare de lunga durata, dar dispunem de spatiu de depozitare limitata.

- Sistemul de buffering este foarte util in cazul sistemelor de operare de tip multitasking in care avem o varietate de dispozitive I/E si o varietate de procese.

- Gigi este şoferul unui TIR de fabricaţie nemţească, anul 2009. În timp ce se află în Anglia, în deplasare de serviciu, Gigi trece pe lângă un semn de STOP, dar fără să oprească. După aceea, Gigi o ia la stânga deşi semnul de circulaţie îi interzicea acest lucru. Făcând asta, Gigi observă că merge într-o direcţie greşită, pe o stradă cu sens unic. Într-un final, Gigi realizează că merge pe partea dreapta a drumului, şi chiar dacă a trecut pe lângă o maşină de poliţie, poliţiştii nu l-au oprit. Totuşi, Gigi nu a încălcat nicio regulă de circulaţie. Cum este posibil ?