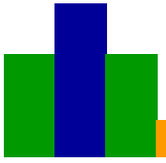


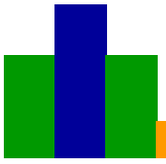
Lecţia 14:

Interfata: procesor - periferice II

G Stefănescu — Universitatea Bucureşti

Arhitectura sistemelor de calcul, Sem.1
Octombrie 2016—Februarie 2017

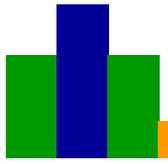
După: D. Patterson and J. Hennessy, Computer Organisation and Design



Interfata: procesor - periferice

Cuprins:

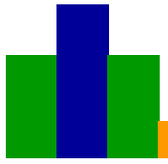
- Generalitati
- Performanta
- Tipuri de dispozitive I/O
- *Magistrale*
- Interfata: I/O - procesor/memorie/OS
- Concluzii, diverse, etc.



Magistrale

Generalitati:

- Comunicarea între diverse componente (procesor, memorie, dispozitive I/O, etc.) se face folosind *magistrale (bus-uri)*;
- Magistralele sunt linii de comunicare pe care le folosesc *în comun* mai multe componente;
- Ele sunt *ieftine, flexibile*, şi permit *adăugarea de noi componente* la sistemul de calcul;
- Limitatea principală: *"gâtuire comunicaţională"*, i.e., limitează viteza de comunicare între componente la cea a bus-ului.
- Este dificil de obţinut viteze mari:
  - motive fizice (lungime fire, număr componente);
  - contradicţii între obiective: lărgimea mare de bandă dă viteză mare, dar creşte timpul de acces la magistrală;



..Magistrale

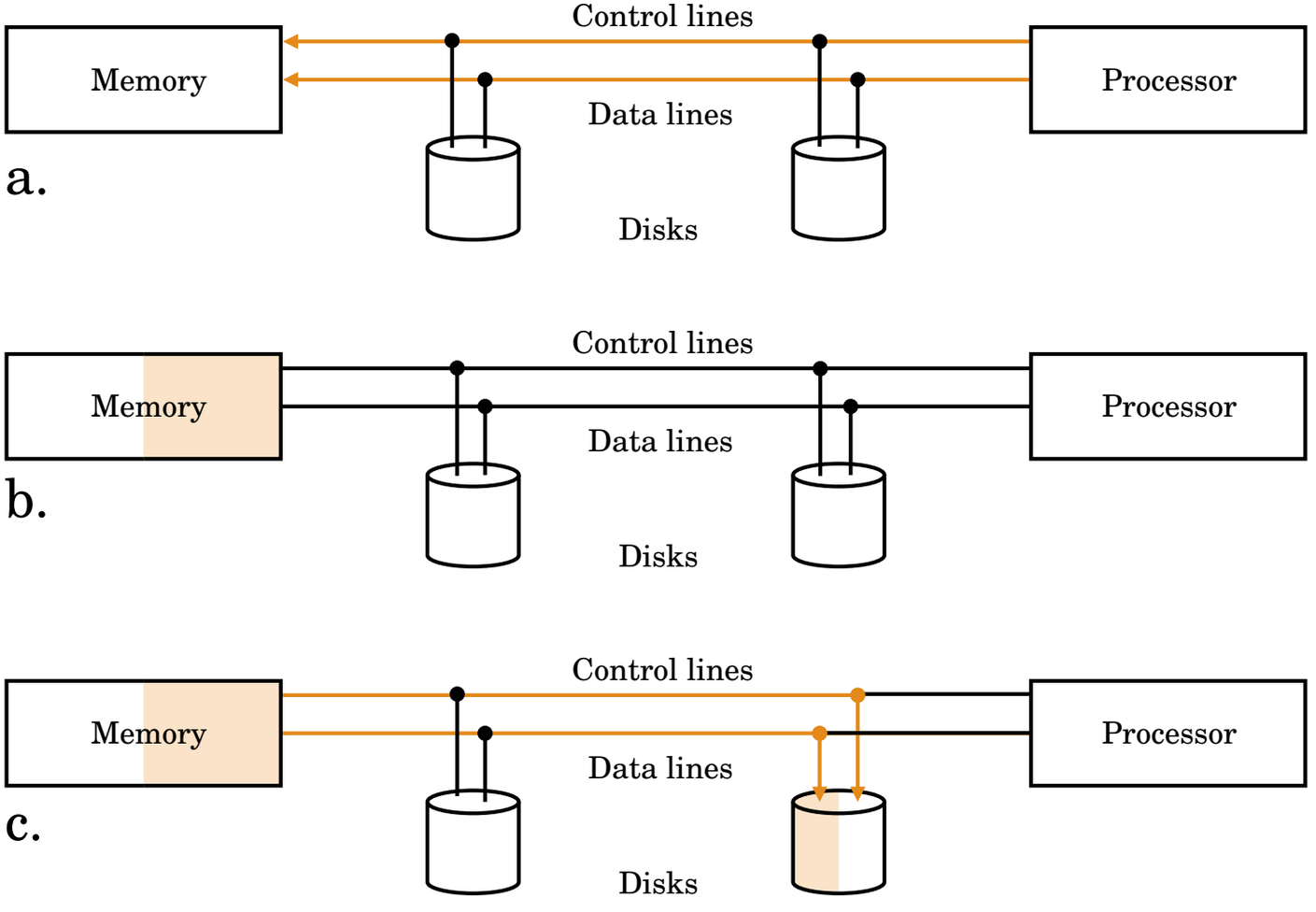
Generalitati (cont.)

- O magistrală are *linii de control* şi *linii de date*:
  - liniile de control sunt pentru semnale de cerere, confirmări, ori specifică tipul informaţiei;
  - liniile de date transferă informaţia "sursă $\mapsto$ destinaţie"
- Operaţia de bază este o *tranzacţie de magistrală* şi cuprinde 2 parţi:
  - transmisie de adresă;
  - transfer de date:
    - *de intrare* (dinspre dispozitive I/O spre memoria procesorului);
    - *de ieşire* (dinspre memoria procesorului spre dispozitive I/O);

..Magistrale

Figura descrie cei 3 paşi la accesări de *ieşire*:

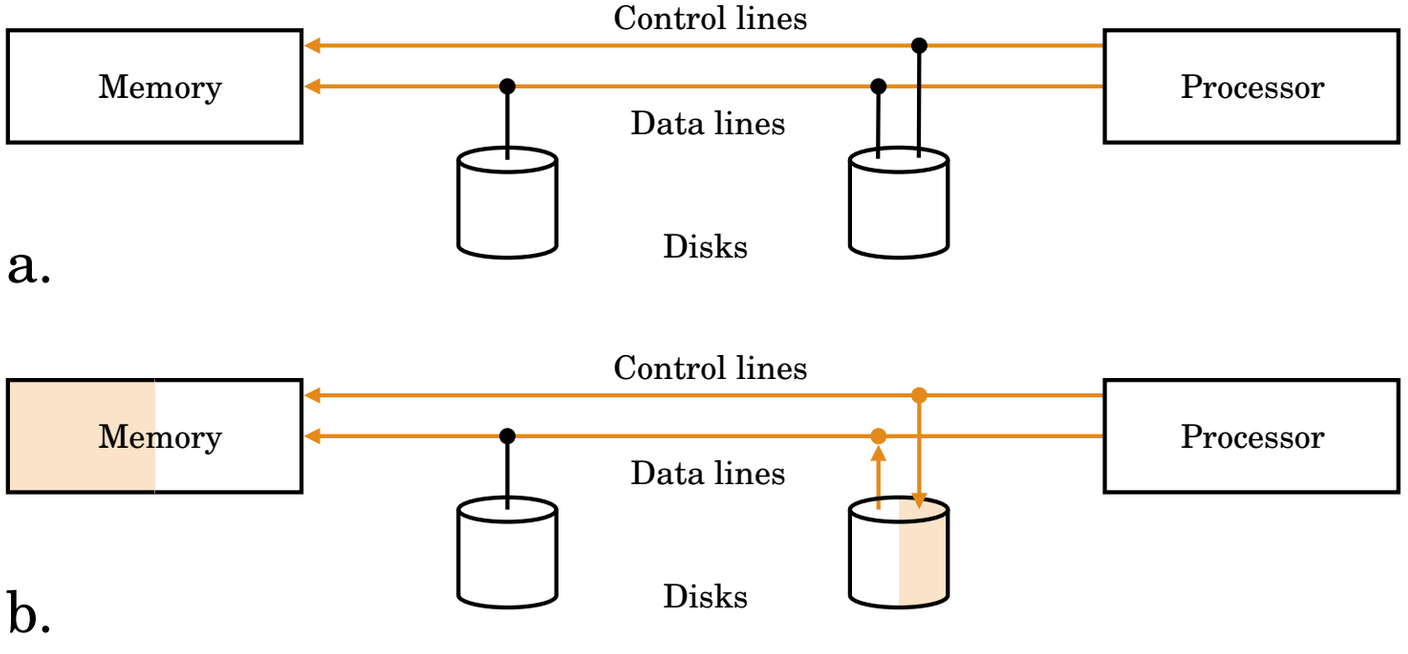
- (a) iniţierea citirii din memorie (control,date)=(cerere,adresă);
- (b) accesarea memoriei pentru citit;
- (c) transfer de date pe magistrală (control,date)=(ack,date).



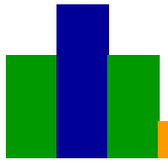
..Magistrale

Figura descrie cei 2 paşi la accesări de *intrare*:

- (a) iniţierea scrierii în memorie (control,date)=(cerere,adresă);
- (b) transfer de date pe magistrală (control,date)=(ack,date).



Notă: Ca în desenele de la procesor, codul de haşurare este: stânga=scriere; dreapta=citire.



Tipuri de magistrale

Tipuri de magistrale

Magistrale procesor-memorie:

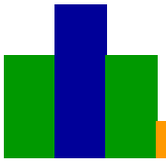
- între procesor şi memorie; specializate;
- scurte şi rapide;

Magistrale I/O:

- pot avea multe dispozitive I/O conectate; universale;
- lungi, dar mai lente;
- nu sunt direct conectate la memorie, ci prin magistrale procesor-memorie, ori magistrale “backplane”

Magistrale backplane (originar folosite la placa de bază):

- permit acces simultan la procesor, memorie, dispozitive I/O;
- sunt reutilizabile (bune pentru diverse configuraţii)



..Tipuri de magistrale

Tipuri de magistrale (cont.)

- Magistralele I/O necesită o interfaţă simplă, pe când cele "backplane" necesită logică suplimentară la interfaţă;
- Avantajul magistralelor "backplane" este că conţin singure toate conexiunile;
- Se pot crea conexiuni complexe ca în figura ce urmează, (a)-(c) (în cazul folosirii de multiple tipuri de magistrale, trebuiesc utilizate *adaptoare de magistrale* pentru a le interconecta)
  - (a) un singur bus backplane: folosit la PCuri vechi;
  - (b) un bus procesor-memorie PCI (bus backplane) care are conexiuni SCSI cu bus-uri I/O: folosit la PCuri noi;
  - (c) ca în (b) + un bus separat pentru traficul procesor-memorie: folosit la multiprocesoare SiliconGraphics

..Tipuri de magistrale

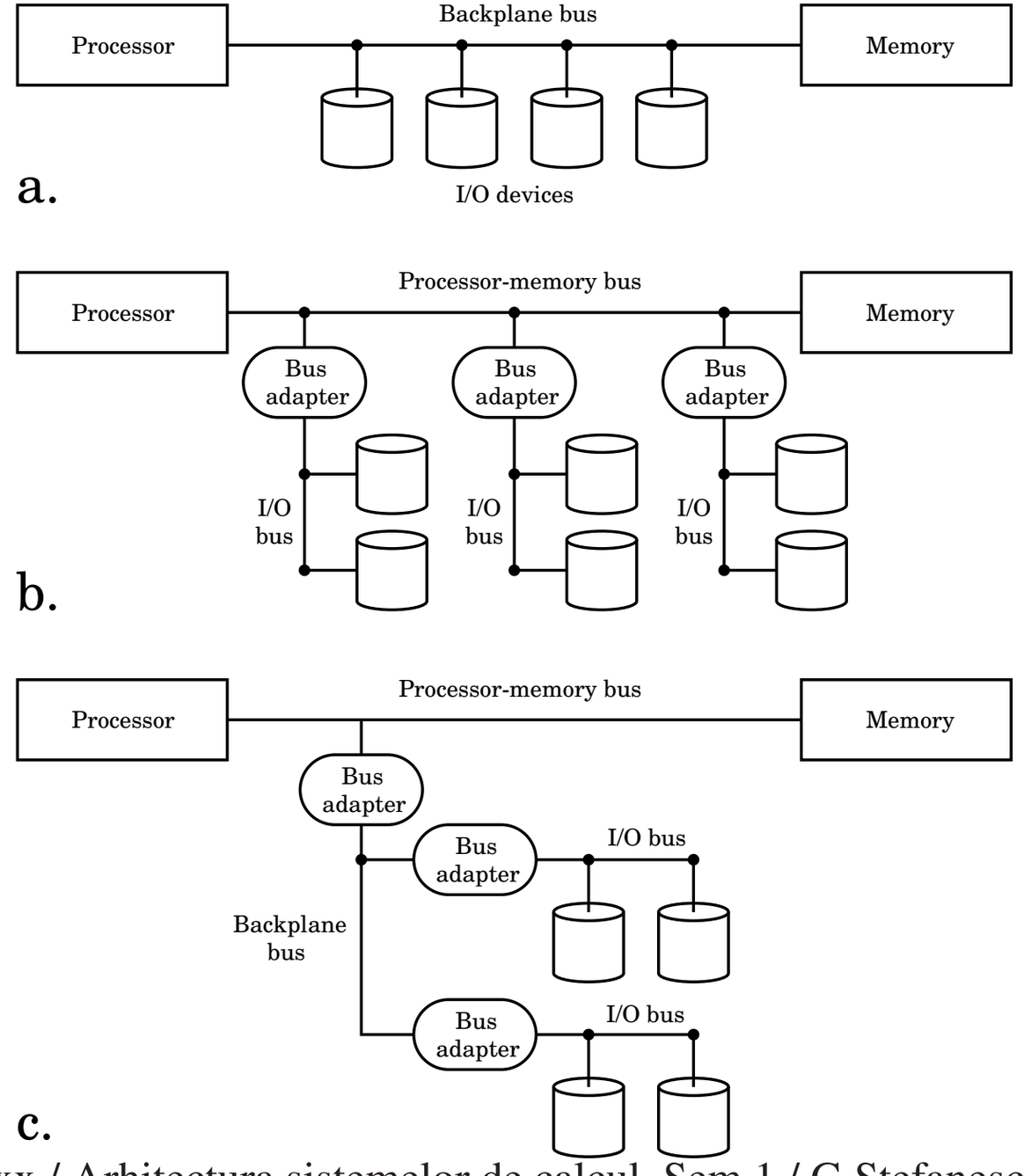
Tipuri de magistrale (cont.)

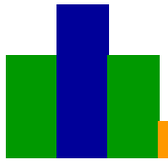
Figura conţine *configuraţii cu magistrale* pentru comunicarea procesor-memorie şi pentru trafic I/O:

(a) - doar magistrală *backplane*;

(b) - magistrală de tip *procesor-memorie* + *magistrale I/O*;

(c) - *toate tipurile*



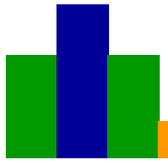


Magistrale sincrone si asincrone

Generalitati: Cele 3 tipuri de magistrale pot folosi diverse scheme de comunicare, distincţia majoră fiind dată de *sincronicitate*.

Comunicare sincronă:

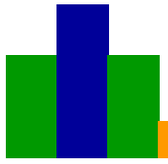
- Liniile de control *au ceas* incorporat, iar magistrala foloseşte un protocol dependent de ceas;
- *Protocolul este simplu* de specificat, folosind o maşină cu stări finită: e.g., în ciclul 1 citeşte adresa, în ciclul 5 scrie date, etc;
- Dezavantaje majore: (1) fiecare dispozitiv trebuie să opereze la *acelaşi ceas*; (2) din cauza *abaterilor de timp* pe frontul de undă, magistrala *nu* poate fi prea *lungă*;



Magistrale asincrone

Comunicare asincronă:

- La magistralele cu protocoale asincrone *nu există ceas*, mărind varietatea de dispozitive I/O conectabile;
- Coordonarea transmisiei se face cu un protocol de *handshaking* (*strângere de mână*);
- Exemplu (interacţie cu memoria): Folosim 3 linii de control:
  1. *ReadReq*: indică o cerere la memorie; simultan se pune adresa pe liniile de date;
  2. *DataRdy*: indică prezenţa cuvântului pe liniile de date spre a fi transferat; (la tranzacţie de intrare este setat de dispozitivul I/O; la tranzacţie de ieşire, este setat de memorie);
  3. *Ack*: confirmare de primire de la partener pentru un semnal ReadReq ori DataRdy;



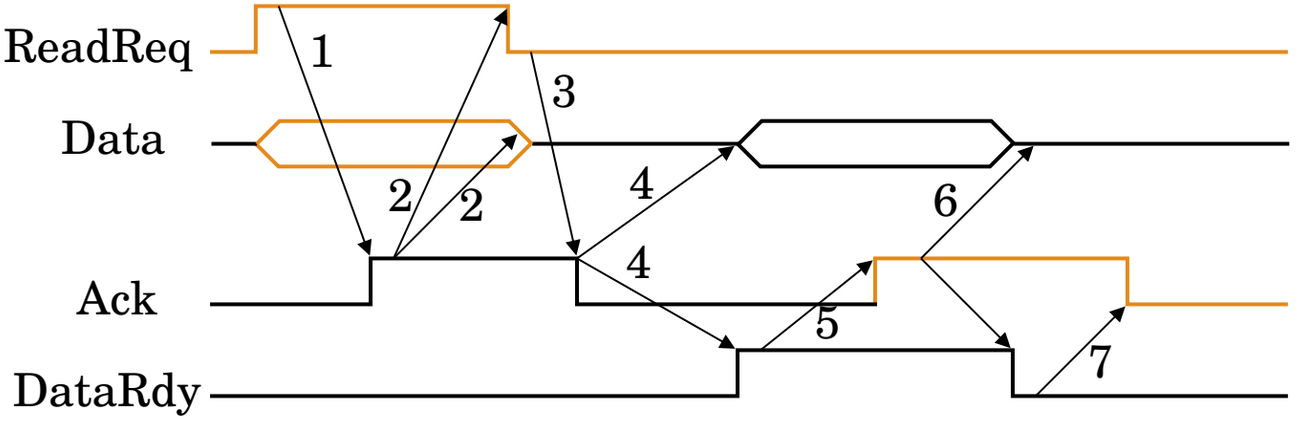
..Magistrale asincrone

Comunicare asincronă (cont.)

- In cazul asincron, semnalele *se ţin setate* până se primeşte confirmarea de la partener (handshaking);
- Protocolul decurge ca la o *pereche de maşini* cu stări finite, fiecare putând trece de la un pas la altul numai după confirmarea partenerului;
- Dacă componentele care comunică au ceas, la interfaţa cu magistrala *semnalul asincron se transformă într-unul sincron* (se digitalizează);
- Pot apare *eşecuri de sincronizare* când avem un semnal incert (între 0 şi 1) şi elementul flip-flop intră într-o stare metastabilă;
- Soluţie - folosim *sincronizatoare* pentru a reduce probabilitatea de eşec;

..Magistrale asincrone

Interogare memorie:

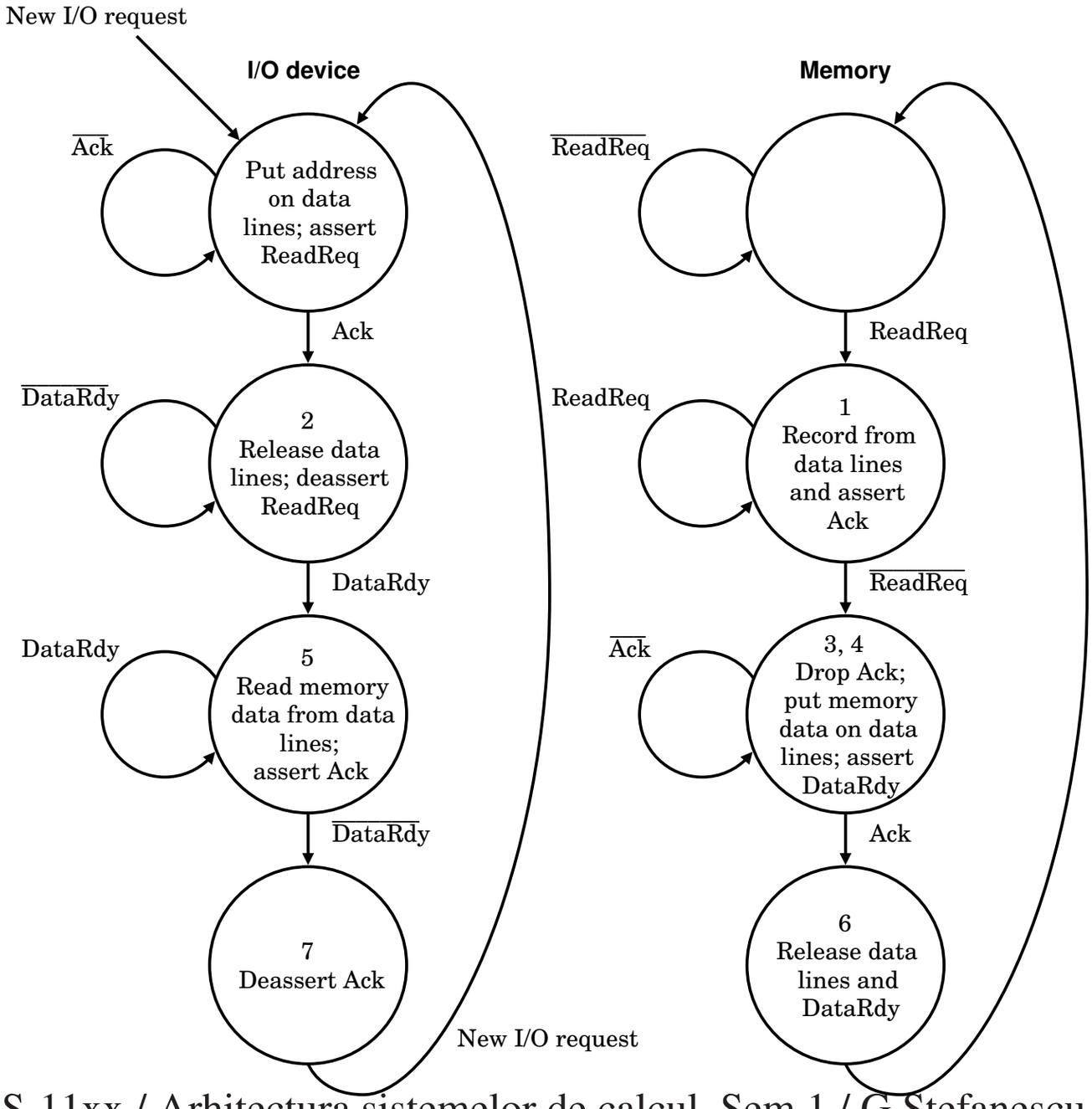


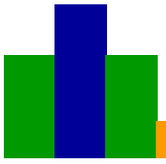
Incepe cu ReadReq setat şi adresa pe liniile de date; Apoi:

1. Memoria vede ReadReq setat, citeste adresa, setează Ack;
2. I/O vede Ack setat, eliberează ReadReq şi liniile de date;
3. Memoria vede ReadReq desetat, desetează Ack;
4. Acces la memorie; Memoria setează DataRdy şi pune datele;
5. I/O vede DataRdy setat, citeşte datele, apoi setează Ack;
6. Memoria vede Ack setat, desetează DataRdy, eliberează liniile de date;
7. I/O vede DataRdy desetat, desetează Ack (end);

..Magistrale asincrone

O pereche de maşini cu stări finite ce implementează protocolul descris verbal mai sus;





Performanta magistralelor

Performanta magistralelor sincrone vs. asincrone:

Problemă: Interogăm o memoria citind cuvinte (acces 200 ns) cu un bus sincron (cicluri de 50 ns, transmisia durează 1 ciclu, lăţime 32b) şi unul asincron (40 ns handshaking, lăţime 32b). Care este viteza de transmisie pe fiecare bus?

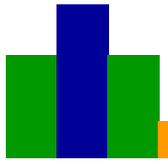
Răspuns:

- Sincron: 50 ns (send adr.) + 200 ns (read) + 50 ns (send data) = 300 ns;

  Viteză (sincron): 4 B/300 ns = 4 MB/0.3 s = 13.3 MB/s;

- Asincron (suprapunem citit memorie cu paşi handshaking): 40 ns (pas 1) + max(3×40 ns, 200 ns) (paşi 2-4, citit) + 3×40 ns (paşi 5-7) = 360 ns;

  Viteză (asincron): 4 B/360 ns = 4 MB/0.36 s = 11.1 MB/s;



..Performanta magistralelor

Cresterea largimii magistralei: Câteva tehnici sunt:

Lăţimea de date: crescând lărgimea liniilor, reducem timpul de transfer;

Adresă separată vs. multiplexată: cu linii separate de adresă, suprapunem trimiterea adresei cu a datelor (scade timpul la scris);

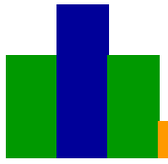
Transfer de blocuri: trimiţând mai multe date odată scade supraîncărcarea datorată stabilirii conexiunii;

..Performanta magistralelor

Cresterea largimii magistralei (cont.)

Split transaction protocol (eliberăm magistrala în periada de acces la memorie)

1. Dispozitivul *I/O* atenţionează memoria transmiţând o *cerere şi adresa*;
2. După *confirmarea primirii* de către memorie, partenerii (dispozitivul I/O şi memoria) *eliberează* liniile de control;
3. Memoria accesează datele, iar *magistrala este liberă* pentru alte comunicaţii;
4. *Memoria* atenţionează dispozitivul I/O indicând că *datele sunt disponibile*;
5. Dispozitivul I/O *primeşte datele*, *confirmă* că le are pentru ca memoria să *elibereze magistrala*;

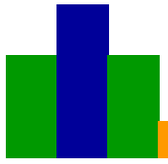


..Performanta magistralelor

Cresterea largimii magistralei (cont.)

Comentarii:

- Pro:
  - *Creşte eficienţa* magistralei, dacă memoria poate satisface cereri suprapuse;
- Contra:
  - Pentru o tranzactie, magistrala trebuie *accesată de 2 ori*;
  - Memoria trebuie să "ştie/memoreze" *identitatea dispozitivului* spre a iniţia a 2-a parte a protocolului;



Acces la magistrala

Accesul la magistrala:

- Interacţia dispozitivelor I/O cu magistrala trebuie *coordonată*; e.g., dacă fiecare seteză liniile de control liber, se crează haos.
- Soluţii posibile:

  Un master (procesorul):

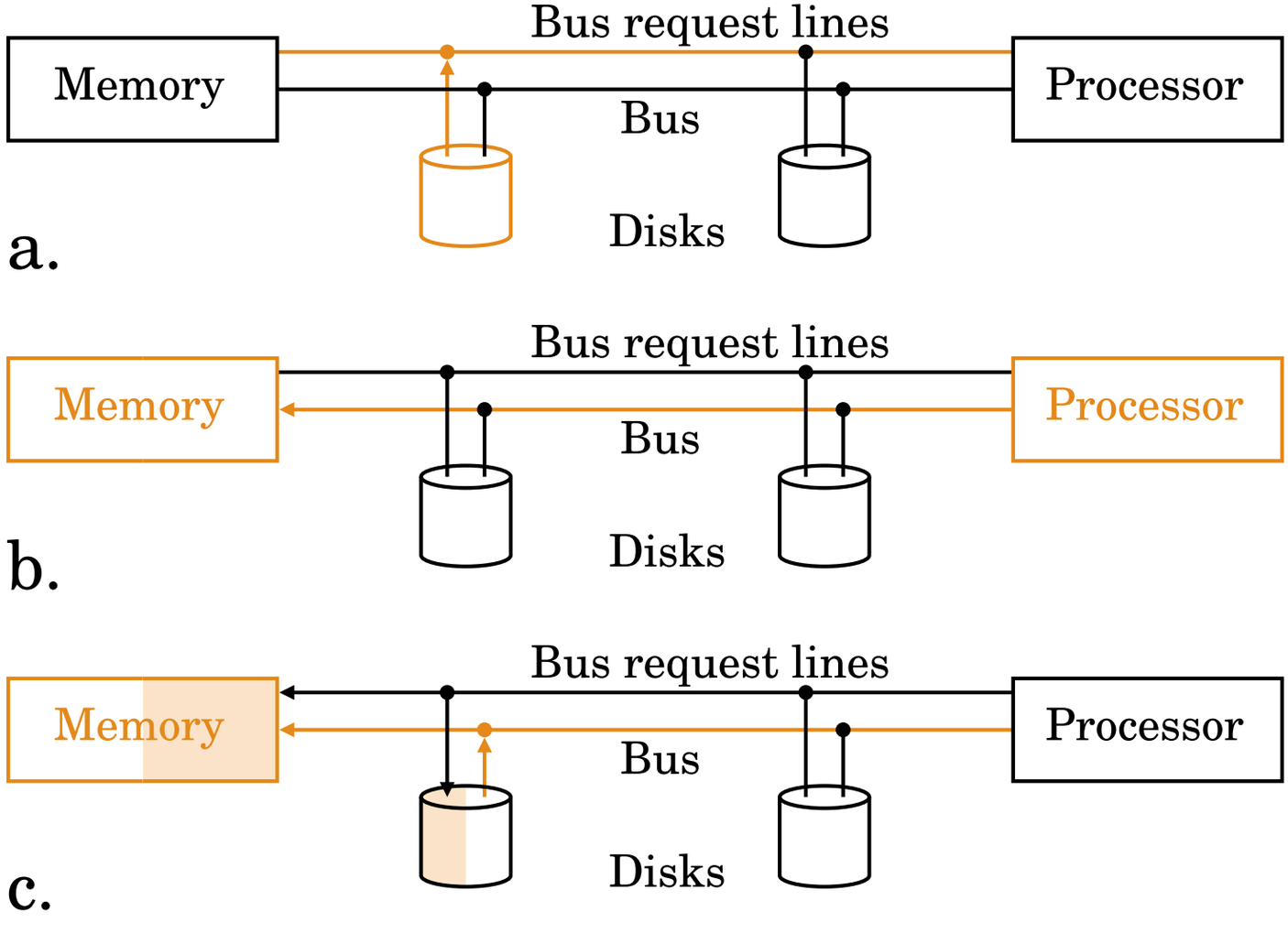
  – OK, dar se gâtuie activitatea procesorului;

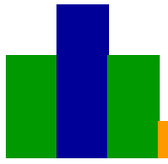
  Mai mulţi master-i:

  – OK, dar se pune problema *arbitrajului*:
  Cum se decide cine primeşte accesul?

..Acces la magistrala

Exemplu: Un *unic master* (procesorul); paşii iniţiali arată ca în figură.

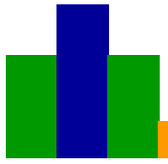




..Acces la magistrala

Arbitru:

- In cazul în care există mai mulţi master-i, trebuie folosită o schemă (protocol), numită *arbitrare a magistralei* care să decidă cine o poate folosi.
- Câteva condiţii care trebuiesc satisfăcute:
  - Dispozitivele *cer access* la magistrală şi ulterior *primesc accesul*; după folosire *eliberează* magistrala;
  - Alocarea acceselor la magistrală se face pe baza unor *priorităţi*;
  - Schema de alocare trebuie să fie *onestă* (fair): orice dispozitv care cere accesul îl va primi cândva;



..Acces la magistrala

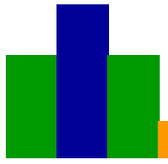
Scheme de arbitrare:

Daisy chain arbitration (arbitrare secvenţială): se folosesc dispozitive puse *în linie după prioritate*;

Centralized, parallel arbitration (arbitrare paralelă centralizată): există un *arbitru central* care decide cine poate folosi magistrala;

Distributed arbitration by self-selection (arbitrare distribuită cu auto-selectare): un dispozitiv se *auto-alege*, dacă este *cel mai prioritar* dintre cele ce solicită acccesul;

Distributed arbitration by collision detection (arbitrare distribuită cu detectarea coliziunilor): *oricine* poate folosi magistrala, ulterior *detectându-se eşecurile* (coliziuni);



..Acces la magistrala

Scheme de arbitrare:

Daisy chain arbitration (arbitrare secvenţială):

- Alocarea se face în linie de la cel mai prioritar la cel mai putin prioritar dispozitiv;
- Un semnal de alocare este propagat şi reţinut de cel mai prioritar dispozitiv (care solicită acces);
- Pro: protocolul este simplu;
- Contra: probleme legate de onestitate (un proces puţin prioritar riscă să-i fie amânat accesul la nesfârşit);

..Acces la magistrala

Scheme de arbitrare (cont.)

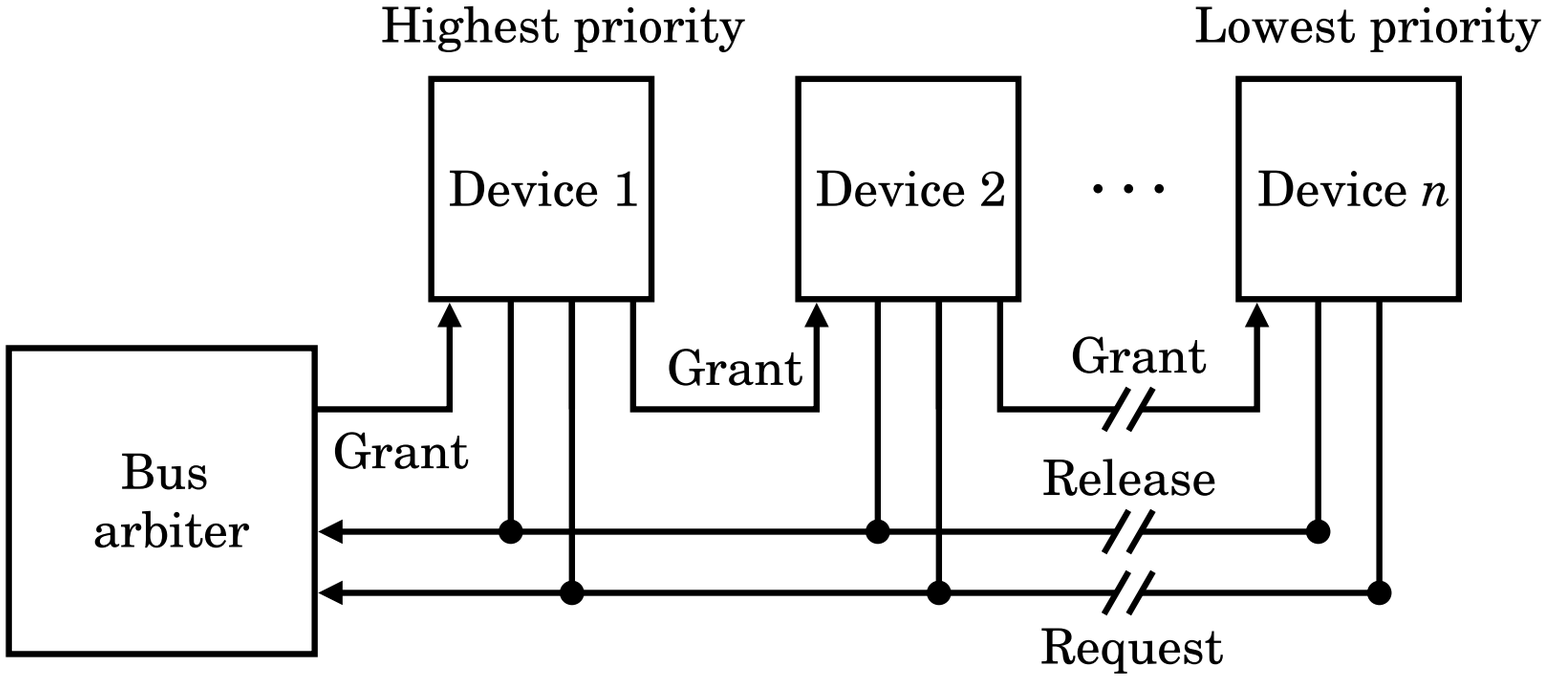
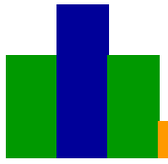


Figura prezintă *"daisy chain arbitration"*, dispozitivele fiind aliniate pe o linie in funcţie de priorităţi, de la cea mai mare la cea mai mică. (Numele "daisy chain" vine de la această aliniere în linie a dispozitivelor.)

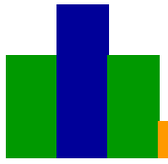


..Acces la magistrala

Protocol (daisy chain arbitration):

1. Semnalizează pe linia de *cerere* (Request);
2. Vezi, pe linia de alocare (Grant) *de la mic la mare*, dacă nu-s tranziţii şi magistrala este *liberă* pentru realocări (evită preluarea magistralei de la cei mai mici)
3. Interceptează semnalul de *alocare* (Grant) şi *nu* lăsa dispozitivele cu prioritate *mai mică* sa-l vadă; *desetează* semnalul pe linia de *cerere*;
4. *Utilizează* magistrala;
5. Semnalizează că magistrala este liberă setând linia de *eliberare* (Release);

Fairness: Pentru onestitate, se poate folosi schema "un dispositiv care a folosit magistrala poate seta linia de cerere doar dupa ce este desetata".

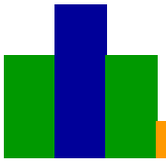


..Acces la magistrala

Scheme de arbitrare (cont.):

Centralized, parallel arbitration (arbitrare paralelă centralizată):

- Există *mai multe linii* de cerere;
- Un *arbitru central* analizează cererile şi alocă magistrala anunţând dispozitivele selectate;
- Exemplu: magistrala “backplane” PCI;
- Contra: cu arbitru central se poate *gâtui* comunicarea pe magistrală;

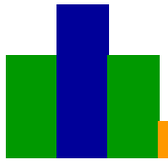


..Acces la magistrala

Scheme de arbitrare (cont.):

Distributed arbitration by self-selection (arbitrare distribuită cu auto-selectare):

- Există mai multe linii de cerere;
- Fiecare dispozitiv care cere accesul se uită la toate care cer accesul şi se autoselectează dacă are cea mai mare prioritate;
- Exemplu: magistrala “backplane” NuBus (de la Apple Macintosh);

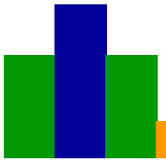


..Acces la magistrala

Scheme de arbitrare (cont.):

Distributed arbitration by collision detection (arbitrare distribuită cu detectarea coliziunilor)

- Fiecare dispozitiv cere independent accesul la magistrală;
- Dacă mai multe dispozitive cer simultan magistrala se produce o coliziune;
- Există un mecanism de detectare a coliziunilor şi selectare a unui dispozitiv pentru comunicare;
- Exemplu: magistrala (protocolul) Ethernet;



Standarde de magistrala

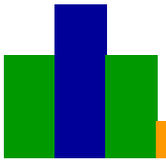
Standarde:

- Au apărut prin efortul unor grupuri interesate (inclusiv din lumea academica); aprobate de ANSI ori IEEE;
- Exemple populare:
  - *PCI* - o magistrală universală de tip “backplane”;
  - *SCSI* (Small Computer Systems Interface) - o magistrală de tip I/O;
- Câteva caracteristici pentru PCI şi SCSI sunt prezentate sumar în tabel:

..Standarde de magistrala

Standarde (cont.)

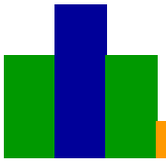
Caracteristici	PCI	SCSI
Tip	backplane	I/O
Latime de date (standard)	32-64	8-32
Adresa & date multiplexate ?	multiplexate	multiplexate
Numar de master-i	multipli	multipli
Arbitrare	paralela, centralizata	auto-selectare
Ceas	sincron, 33-66 MHz	asincron ori sincron (5-10MHz)
Banda maxima teoretica	133-512 MB/s	5-40 MB/s
Banda realizata practic	80 MB/s	2.5-40 MB/s (sinc.) 1.5 MB/s (asinc.)
Numar maxim de despozitive	1024 ($\leq$ 32/segment)	7-31 (= largime -1)
Lungime maxima	0.5 m	2.5 m
Nume standard	PCI	ANSI X3.131



Interfata: procesor - periferice

Cuprins:

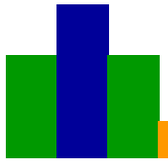
- Generalitati
- Performanta
- Tipuri de dispozitive I/O
- Magistrale
- *Interfata: I/O - procesor/memorie/OS*
- Concluzii, diverse, etc.



Interfata: I/O - procesor/memorie/OS

Generalitati:

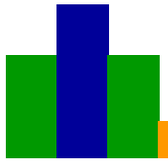
- Gestionarea dispozitivelor I/O necesită răspunsuri la câteva întrebări tipice:
  - Cum se translatează cererile I/O utilizator în comenzi specifice dispozitivului şi cum se transmit la dispozitiv?
  - Cum se transferă datele de la ori spre memorie?
  - Ce rol are OS (sistemul de operare)?



..Interfata: I/O - procesor/memorie/OS

Generalitati (cont.)

- De notat că:
  - *Multiple programe* solicită dispozitive I/O prin intermediul procesorului;
  - Multe dispozitive I/O utilizează întreruperi care folosesc modul *kernel (superuser)*, deci sunt gestionate de OS;
  - Controlul la nivel jos al dispozitivul I/O este în genere *complex*, necesitând manipularea multor evenimente (task-uri) concurente;
- Pentru a garanta accesul la resursele alocate fiecărui user, *OS* trebuie să *comunice cu dispozitivele I/O* şi să *interzică accesul direct* al programului utilizator la dispozitivul I/O.



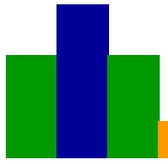
..Interfata: I/O - procesor/memorie/OS

Controlul dispozitivelor I/O:

Pentru a gestiona dispozitivul, procesorul trebuie să-i trimeată comenzi. Sunt preferate 2 metode:

Instrucţiuni I/O speciale:

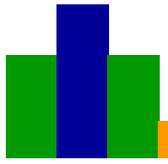
- Există *comenzi speciale*, trimise prin fire de la procesor la dispozitivele I/O (pe liniile de date);
- Instrucţiunile necesită *modul superuser (kernel)*, deci nu-s acceptate în program utilizator;
- Exemple: Intel 80×86, IMB 370;



..Interfata: I/O - procesor/memorie/OS

Comenzi I/O în memorie - *spaţiul de adrese* are *zone dedicate* dispozitivelor *I/O*, iar citirile/scrieriile acolo sunt interpretate drept comenzi:

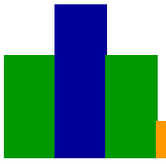
- O scriere în zona I/O este neglijată de memorie (adresa indică o interacţie cu un dispozitiv I/O);
- Unitatea de control a dispozitivului vede operaţia şi preia zona de date ca pe o *secvenţă de comenzi* pentru dispozitivul I/O;
- Accesul direct al utilizatorilor la dispozitivul I/O este *interzis de OS* care-i nu le permite accesarea spaţiului de aderese desitinat pentru I/O;



..Interfata: I/O - procesor/memorie/OS

- Un read/write necesită mai multe operaţii I/O, iar procesorul trebuie să chestioneze dispozitivul I/O spre a şti când a *completat o comandă*;
- Exemplu: Printerul online DEC LP11 are 2 regiştri I/O:
  - unul *de stare* (Status Register) cu:
    - un *done-bit*, care spune dacă a fost printat un caracter;
    - un *error-bit*, dacă nu am hârtie ori e boţită
  - unul *de date* (Data Register) - indică ce se printează;

  Procesorul trebuie să urmărească scrierea caracter-cu-caracter!



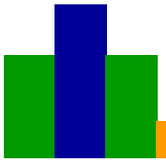
..Interfata: I/O - procesor/memorie/OS

Comunicarea cu procesorul:

Proesorul trebuie să verifice periodic statului dispozitivelor I/O. Din nou, sunt preferate 2 metode:

Interogare periodică (polling):

- Dispozitivul I/O pune informaţii în *registrul de stare*, care este *citit periodic* de procesor;
- Procesorul controlează complet dispozitivul I/O;
- Dezavantaj major: consumăm din timpul procesorului
  - la un dispozitiv *lent* (e.g., mouse), consumul poate fi *acceptabil* - 0.002%;
  - la unul dispozitiv *rapid* (e.g., hard-disk), consumul poate fi *prea mare* - 20%;



..Interfata: I/O - procesor/memorie/OS

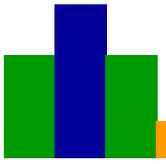
Rata utilizarii procesorului la interogare periodca I/O:

Problemă: Presupunem că o operaţie de polling (activare rutină polling + accesare I/O + restart program user) durează 400 cicli, iar procesorul are 500 MHz.

Care este rata de utilizare a procesorului la interograrea periodică a dispozitivelor I/O de mai jos?

1. Mouse - accesat de 30 ori/sec;
2. Floppy disk - transferând câte 2 B, cu rata 50 KB/s;
3. Hard disk - transferând câte 16 B, cu rata 4 MB/s;

Răspuns:



..Interfata: I/O - procesor/memorie/OS

Rata utilizarii procesorului la interogare periodca I/O:

- Mouse:

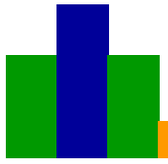
  Cicluri = $30 \times 400 = 12\ 000$;
  Rata = $12\ 000/(500 \times 10^6) = 0.002\%$

- Floppy disk:

  Rata accese = (50 KB/s)/(2B/acces) = 25 K-accese/s;
  Cicluri = $25\text{K} \times 400 = 10^7$;
  Rata = $10^7/(500 \times 10^6) = 2\%$

- Hard disk:

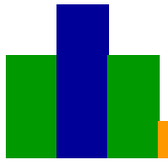
  Rata accese =(4 MB/s)/(16 B/acces) = 0.25 M-accese/s;
  Cicluri = $0.25\text{M} \times 400 = 10^8$;
  Rata = $10^8/(500 \times 10^6) = 20\%$



..Interfata: I/O - procesor/memorie/OS

Intereruperi:

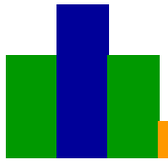
- In loc de a urmări constant dispozitivele I/O, procesorul poate folosi *întreruperi*, anume evenimente emise de dispozitivele I/O când este necesară utilizarea lor;
- Intreruperile I/O sunt *similare cu cele asociate instrucţiunilor* (overflow, instrucţiune inexistentă, etc.), cu 2 diferenţe majore:
  - *asincronicitate:* *nu sunt blocante*, anume pot fi procesate cu o anumită întârziere (când procesorul are timp);
  - *priorităţi:* întreruperile sunt asociate cu diverse dispozitive I/O care au *urgenţe de procesare diferite*;
- Comunicarea întreruperilor lui OS se face cu *întreruperi vectoriale*, ori prin citirea *registrului de cauză* (Cause Register);



..Interfata: I/O - procesor/memorie/OS

Transferarea datelor (Dispozitiv I/O - Memorie):

- Se folosesc tehnicile anterioare: *polling* ori *întreruperi*;
- In cazul întreruperilor, *creşte latenţa de activare* a procedurii, dar *scade ocuparea procesorului* (care controlează dispozitivul I/O doar când acesta este activ);
- Alternativ, se poate folosi schema *DMA - direct memory access* care *eliberează procesorul de controlul detaliat* al dispozitivului I/O:
  - procesorul este acum ocupat doar cu *lansarea* şi *finalizarea* procedurii de acces a dispozitivului I/O;



..Interfata: I/O - procesor/memorie/OS

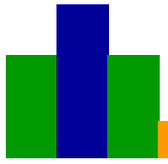
DMA (direct memory access):

1. *Procesorul setează DMA*, indicând: *numele* dispozitivului I/O, *adresa* şi *lungimea* datelor de transfer.

2. *DMA* iniţializează operaţia de *control a dispozitivului* şi *arbitrează bus-ul*:

   - De notat că DMA trebuie să solicite şi să obţină de *mai multe ori accesul* la magistrală pentru a completa întreaga operaţie I/O;
   - In tot acest timp *procesorul este liber*;

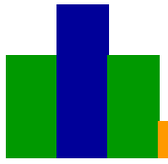
3. Când transferul DMA este complet, se emite un semnal de *întrerupere*; *procesorul* tesează DMA ori memoria spre a verifica *finalizarea* cu success a operaţiei;



..Interfata: I/O - procesor/memorie/OS

DMA - direct memory access (cont.)

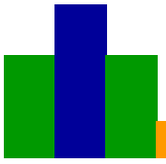
- Intr-un sistem de calcul există *mai multe dispozitive DMA*;
- Exemplu: La un sistem de primul tip (1 magistrală procesor-memorie + multiple magistrale I/O), de regulă unităţile de control I/O între *dispozitivele I/O şi memorie* au procesoare DMA;
- Incărcarea asociată transferurilor DMA este mică:
  - La problema anterioară, dacă iniţializarea + finalizarea DMA costă 1000 + 500 cicluri iar transferul pe/de la hard disk este continu, implicarea procesorului este de 0.2%;
  - De regulă reducerea este mai mică, căci lucrul pe disk nu-i continu, iar DMA gâtuie memoria;
- Dispozitivele I/O pot deveni mai *inteligente*: "procesoarelor" asociate lor li se pot aloca *sarcini complexe*;



..Interfata: I/O - procesor/memorie/OS

Interactia DMA - Sistemul de Memorie: DMA ridică probleme de *coerenţă* în sistemul de memorie:

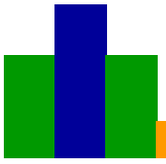
- Prin activitatea DMA, se crează un pericol de incoerenţă între *memoria principală şi cea cache*, ori între *memoria principală şi cea virtuală*;
- Problema se numeşte *problema de coerenţă* ("coherency or stale-data problem");
- Transferul mai mare de o pagină implică translatarea adreselor virtuale în fizice; Soluţii:
  - Se poate da o *mică listă de translaţii* la inţializarea DMA;
  - Se poate folosit tehnica de *serializare* ("chained transfers"): se transferă *pagini, dar* se fac *multiple transferuri* (gestionate de procesorul DMA ori OS);



Interfata: procesor - periferice

Cuprins:

- Generalitati
- Performanta
- Tipuri de dispozitive I/O
- Magistrale
- Interfata: I/O - procesor/memorie/OS
- *Concluzii, diverse, etc.*



Concluzii, diverse, etc.

A se insera...