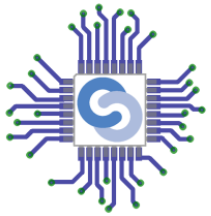
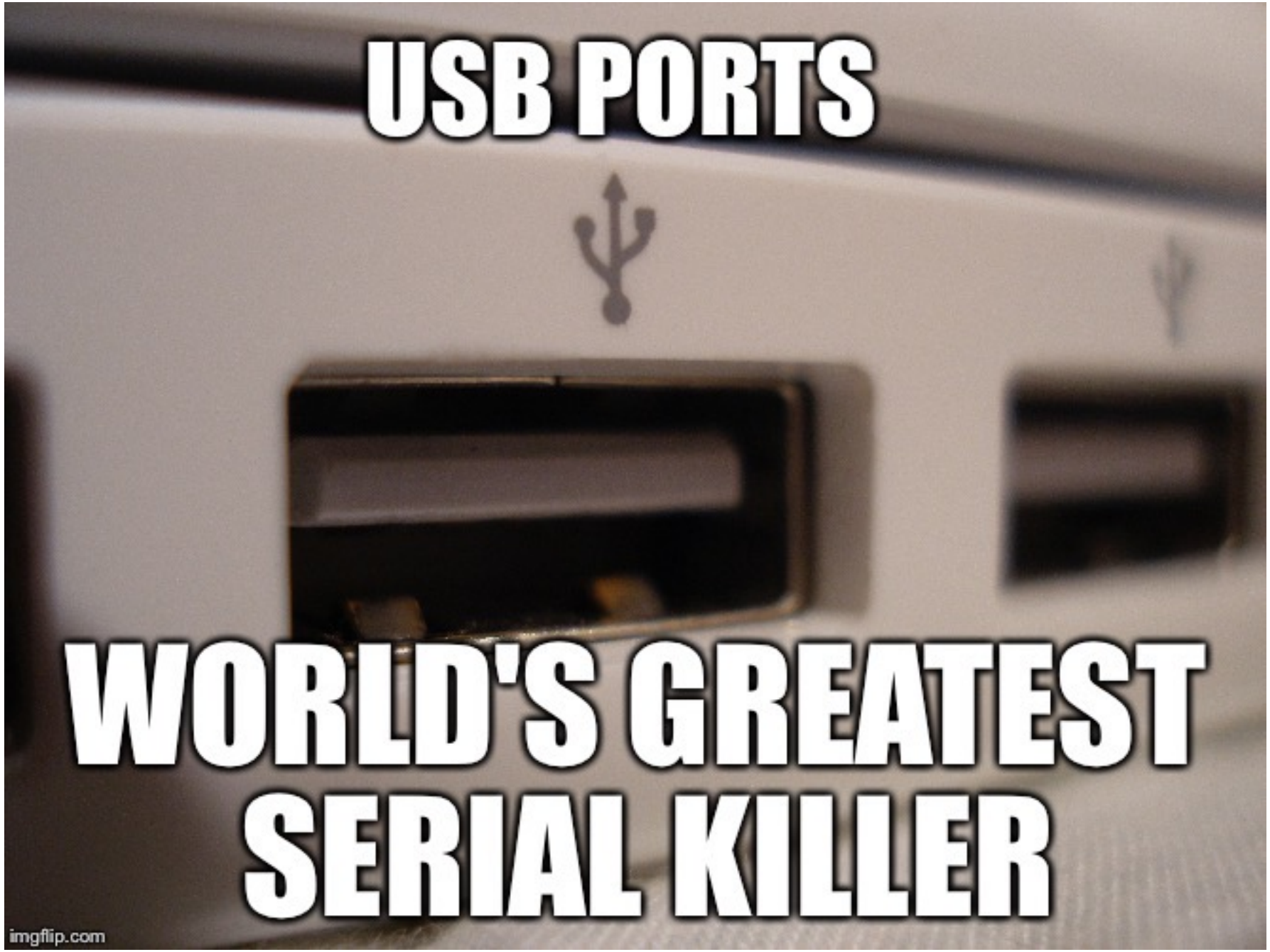


Calculatoare Numerice (2)

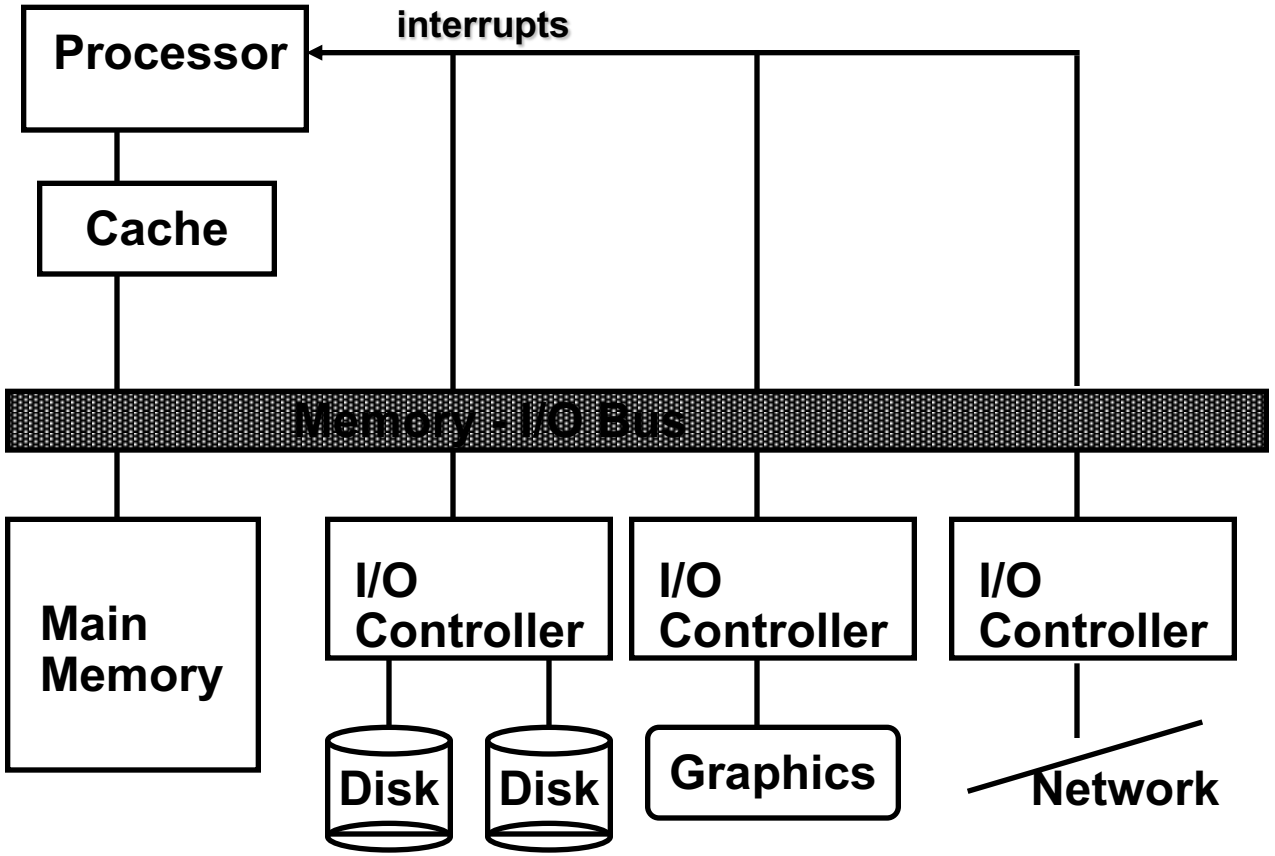
– Cursul 7 –

Dispozitive periferice (2)

Facultatea de Automatică și Calculatoare
Universitatea Politehnica București


USB PORTS
WORLD'S GREATEST SERIAL KILLER
imgflip.com

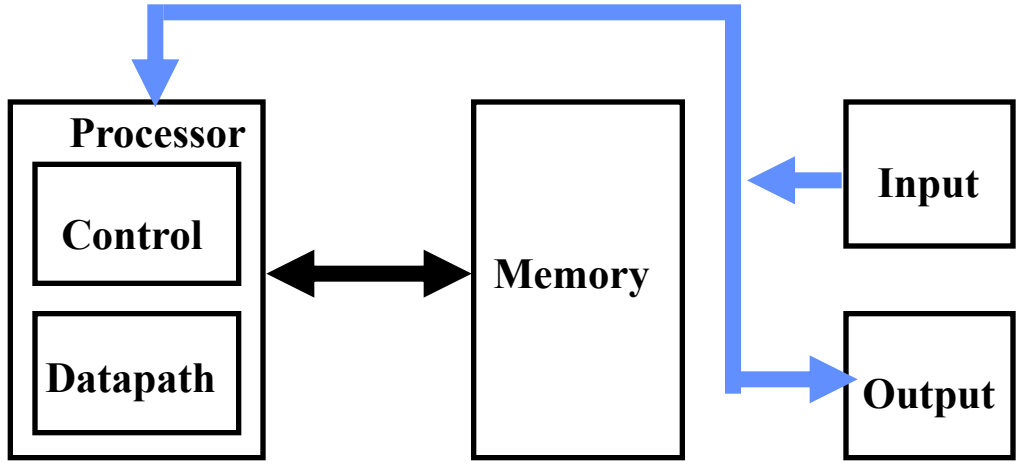
Sisteme I/O


Processor
interrupts
Cache
Memory - I/O Bus
Main Memory
I/O Controller
I/O Controller
I/O Controller
Disk
Disk
Graphics
Network

Ce este o magistrală?

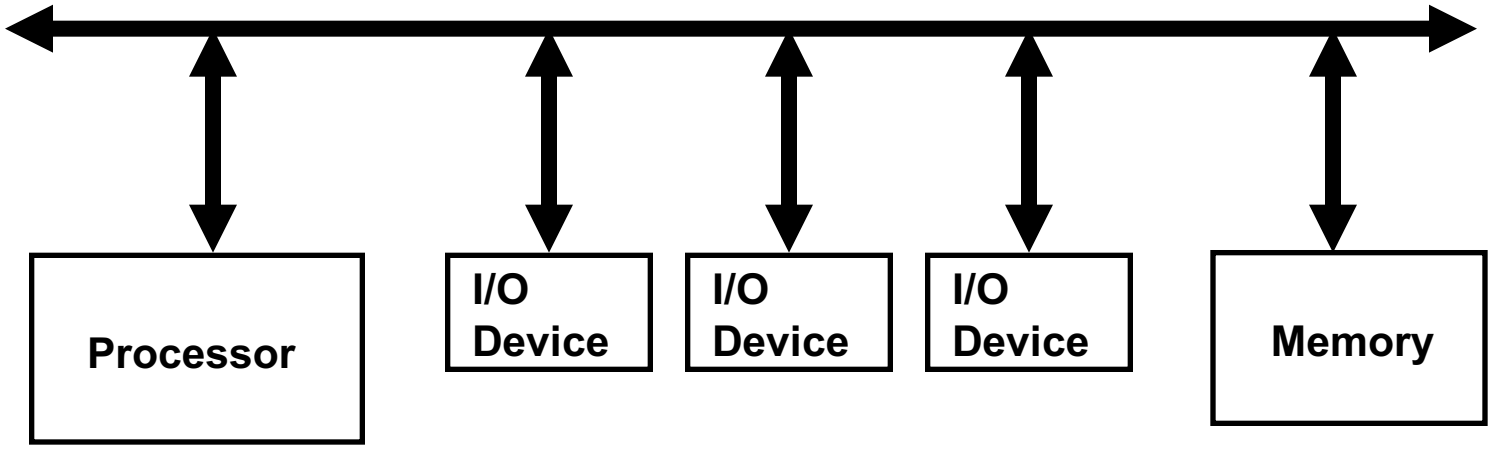
O magistrală este:

- O legătură partajată de comunicație
- Un singur set de linii de legătură folosit pentru a conecta diferite subsisteme



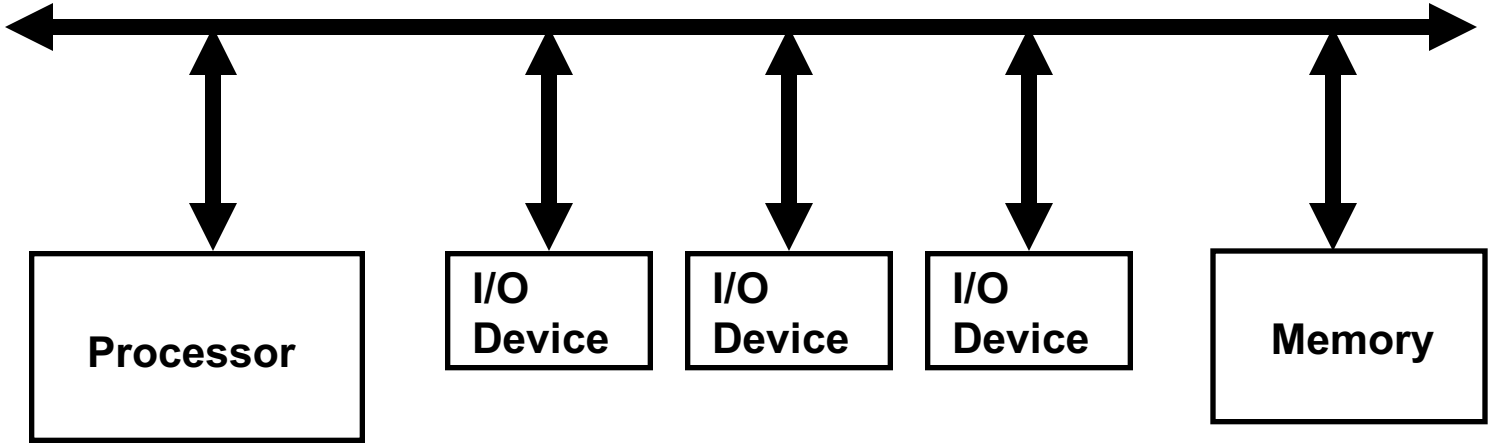
- O magistrală este o componentă vitală în construcția unor sisteme complexe de calcul
  - Metodă sistematică de abstractizare

Avantajele magistralelor



- Versatilitate:
  - Dispozitivele noi pot fi adăugate cu ușurință
  - Perifericele pot fi mutate între diverse sisteme de calcul care respectă același standard
- Cost redus:
  - Un singur set de fire de legătură este partajat de toate perifericele

Dezavantajele magistralelor

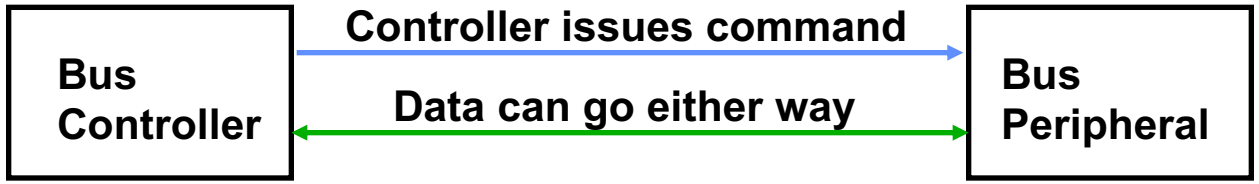


- Creează o gâtuire în comunicație (bottleneck)
  - Lățimea de bandă a magistralei de memorie limitează I/O throughput
- Viteza maximă de transfer de date este limitată de:
  - Lungimea magistralei
  - Numărul de dispozitive de pe magistrală
  - Necesitatea de-a conecta o gamă largă de dispozitive, fiecare cu:
    - Latențe diferite
    - Viteze de transfer diferite

Organizarea generală a unei magistrale

- Linii de control:
  - Signal requests and acknowledgments
  - Indică ce fel de informație circulă pe liniile de date la acel moment
- Linii de date transferă informația de la sursă la destinație:
  - Date și adrese
  - Comenzi complexe

Controller versus Peripheral

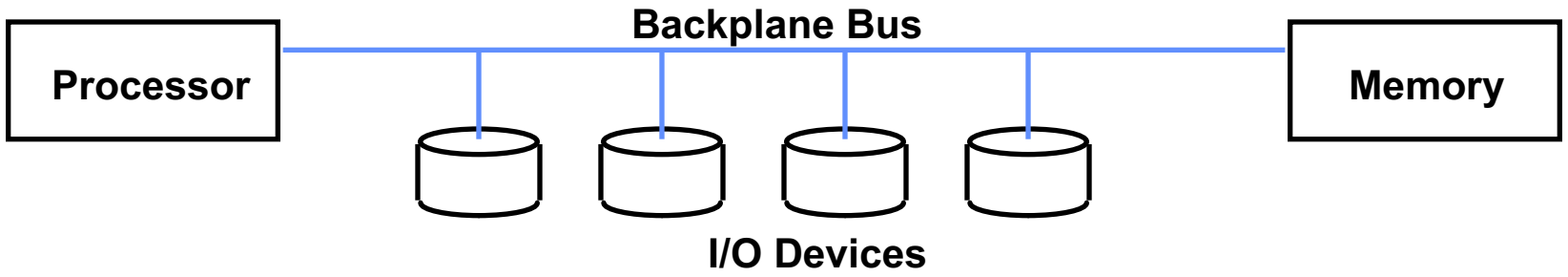


- O tranzacție pe magistrală se face în doi pași:
  - Emiterea comenzii (și a adresei) – cerere
  - Transferul de date – acțiune
- Controller-ul este cel care începe orice tranzacție:
  - Emite comanda (și adresa slave-ului)
- Perifericul este cel care răspunde comenzii:
  - Trimite datele controller-ului, dacă controller-ul cere datele
  - Recepționează date de la controller, dacă controller-ul dorește să trimită datele

Tipuri de magistrale

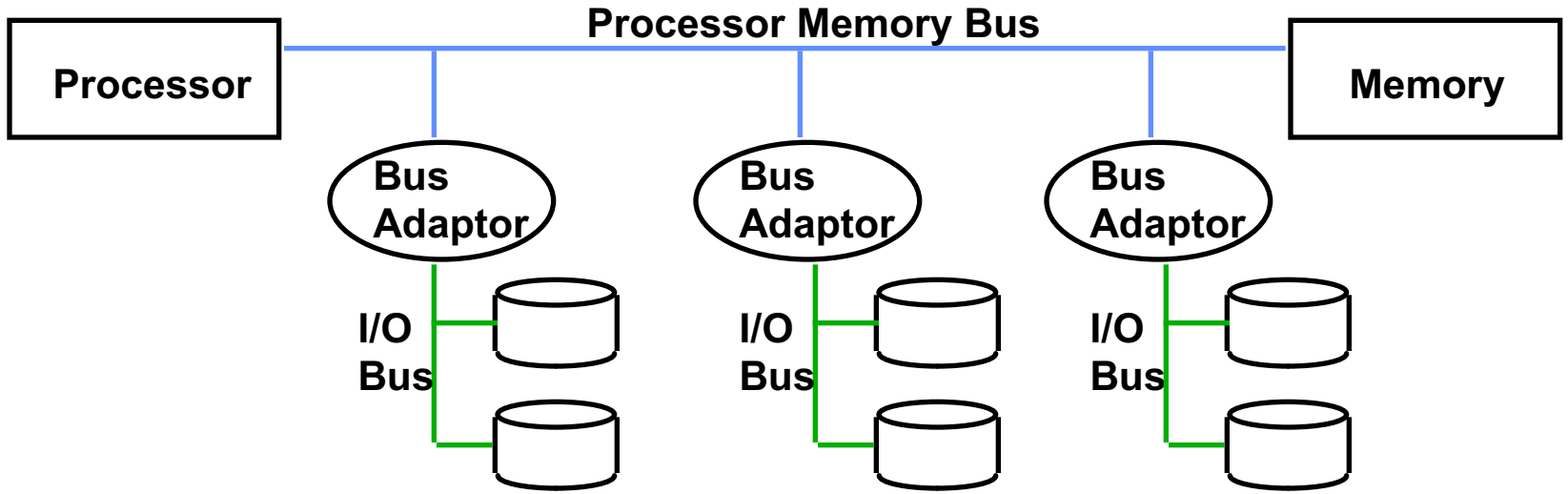
- Magistrală Procesor-Memorie (design specific)
  - Scurtă și de viteză mare
  - Trebuie doar să conecteze sistemul de memorie
    - Maximizează lățimea de bandă memory-to-processor
  - Conectată direct la procesor
  - Optimizată pentru transferuri de blocuri cache
- Magistrală I/O (industry standard)
  - De obicei, de lungime mai mare și mai lentă
  - Trebuie să acomodeze o gamă largă de dispozitive I/O
  - Se conectează la magistrala procesor-memorie sau backplane bus
- Backplane Bus (standard / proprietar)
  - Backplane: o structură de interconectare din interiorul șasiului
  - Permite procesoarelor, memoriei și dispozitivelor I/O să coexiste
  - Avantaje de cost: o magistrală pentru toate componentele
- Magistrale Seriale (tendință generală de interconectare serială)

Sistem de calcul cu o magistrală: Backplane Bus



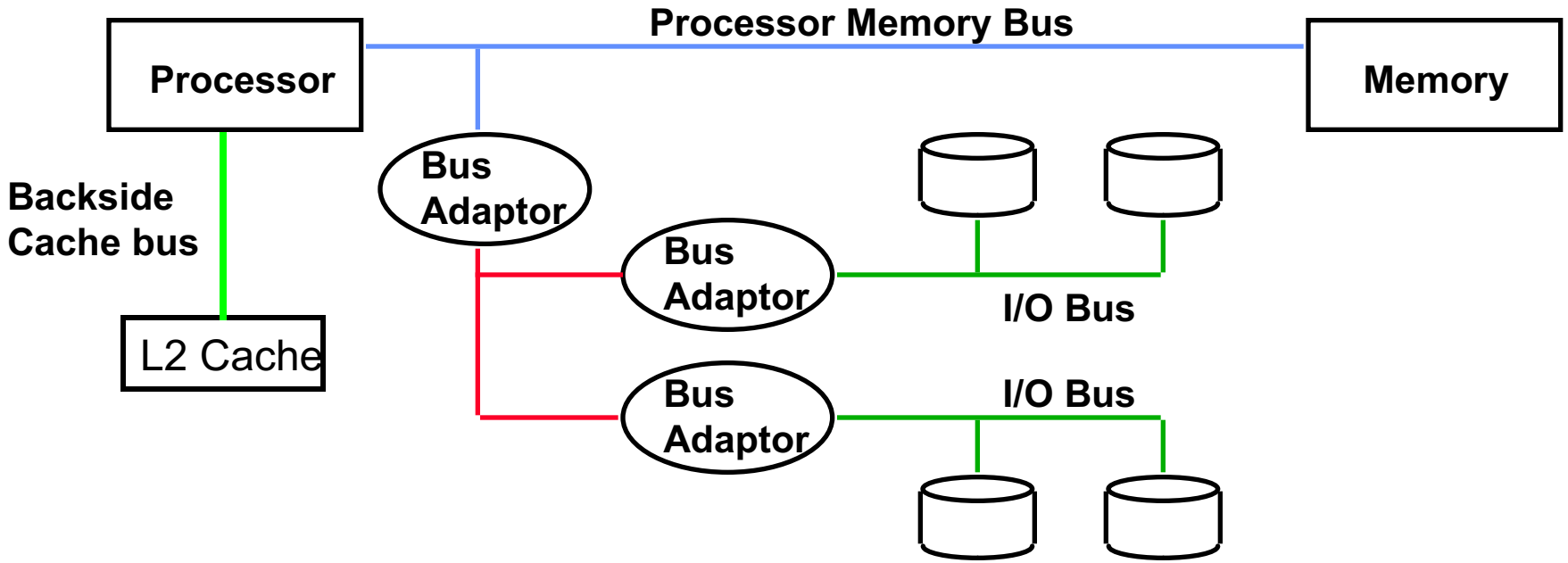
- O singură magistrală (backplane bus) este folosită pentru:
  - Comunicație processor-to-memory
  - Comunicația dintre dispozitivele I/O și memorie
- Avantaje: Simplu și cost redus
- Dezavantaje: lent / magistrala poate deveni un bottleneck
- Exemplu: IBM PC - AT

Sistem cu două magistrale



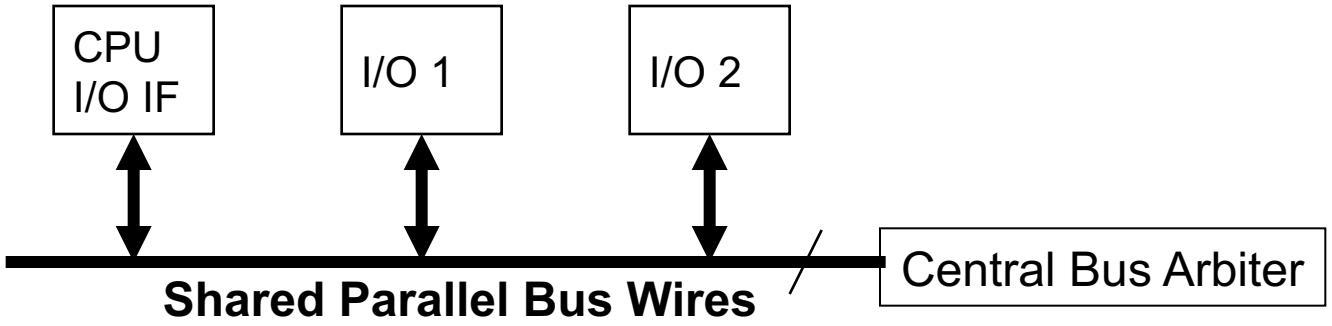
- Magistralele I/O se conectează la magistrala principală prin adaptoare:
  - Processor-memory bus: în special pentru traficul procesor-memorie
  - I/O buses: furnizează conectori de expansiune pentru diferitele dispozitive I/O
- Apple Macintosh-II
  - NuBus: Procesor, memorie și câteva dispozitive I/O
  - SCCI Bus: restul de dispozitive I/O

Sistem cu trei magistrale (+ backside cache)



- Un număr mic de magistrale backplane conectate la magistrala procesor-memorie
  - Magistrala procesor-memorie este optimizată pentru traficul procesor-memorie
  - Magistralele I/O sunt conectate la magistrala backplane
- Avantaj: încărcarea magistralei de procesor este cu mult redusă

Trecerea de la I/O paralel la I/O serial



- Frecvența ceasului pe un bus paralel e limitată de lungimea magistralei (~100MHz)
- Consum de energie mare pentru a conecta un număr mare de periferice
- Central bus arbiter crește latența fiecărei tranzacții
- Conectori scumpi, multe linii de interconectare ce cresc costurile (orice linie în plus adaugă un cost suplimentar)
- Exemple: VMEbus, Sbus, ISA bus, PCI, SCSI, IDE

Linii seriale punct-la-punct dedicate

- Legăturile punct-la-punct funcționează la viteze multi-gigabit folosindcodificări avansate pentru ceas/semnale de date (necesită multă electronică la ambele capete ale comunicației)
- Low power – avem o singură legătură de date cu un singur periferic
- SATA, USB, Firewire, etc.
- Transferuri simultane multiple
- Conectori și cabluri ieftine (mai puțin cupru pentru cabluri/ mai mult siliciu pentru logica de comunicație)
- Fiecare dispozitiv are legătura lui de date, cu parametri proprii de viteză și comunicație
- Exemple: Ethernet, Infiniband, PCI Express,

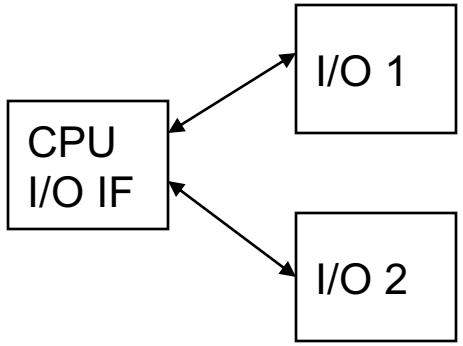
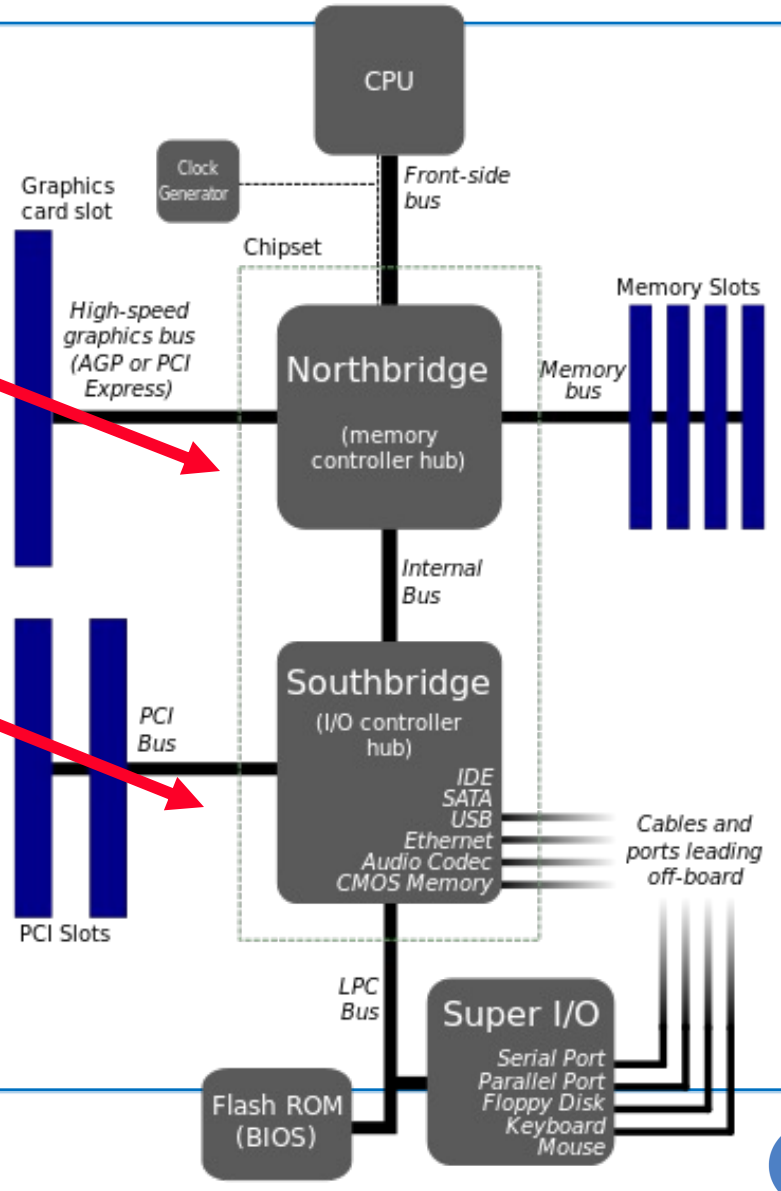


Diagrama interconexiunilor unei plăci de bază

- Northbridge:
  - Handles memory
  - Graphics
- Southbridge: I/O
  - PCI bus
  - Disk controllers
  - USB controllers
  - Audio
  - Serial I/O
  - Interrupt controller
  - Timers



Exemplu: National Semiconductor ns16550 UART

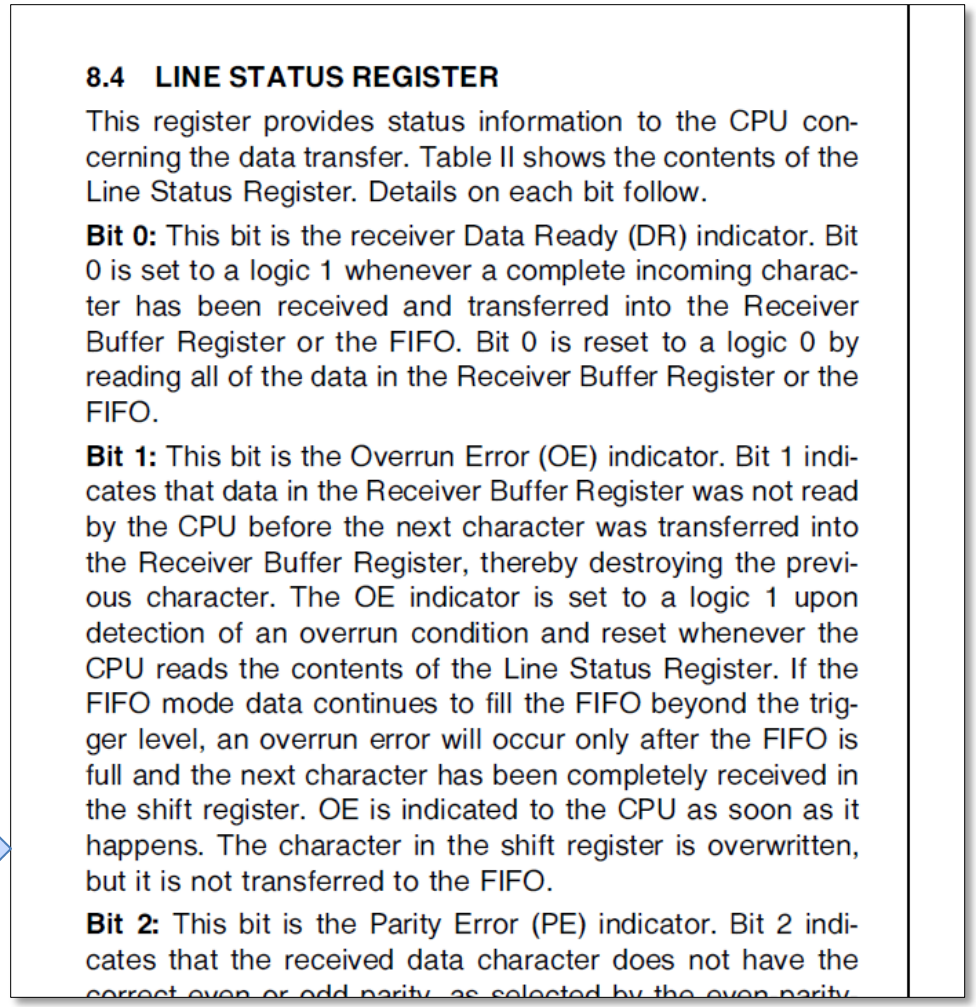
- *Universal Asynchronous Receiver/Transmitter*
  - Dispozitiv serial RS-232
- Port serial IBM PC standard
  - Chip-ul este integrat acum în Southbridge-ul oricărui PC
  - Încă e folosit la servere PC
  - Nu mai e folosit pe laptopuri si foarte rar întâlnit pe desktop-uri în zilele noastre ☹
- Primul device driver care este scris de obicei pentru un SO…


I AM YOUR FATHER

Registre

- Detaliile de implementare pentru registre date în “datasheets”
- Informațiile de acolo sunt câteodată mai puțin adevărate ☺

Din datasheet-ul NS16550 UART



8.4 LINE STATUS REGISTER

This register provides status information to the CPU concerning the data transfer. Table II shows the contents of the Line Status Register. Details on each bit follow.

Bit 0: This bit is the receiver Data Ready (DR) indicator. Bit 0 is set to a logic 1 whenever a complete incoming character has been received and transferred into the Receiver Buffer Register or the FIFO. Bit 0 is reset to a logic 0 by reading all of the data in the Receiver Buffer Register or the FIFO.

Bit 1: This bit is the Overrun Error (OE) indicator. Bit 1 indicates that data in the Receiver Buffer Register was not read by the CPU before the next character was transferred into the Receiver Buffer Register, thereby destroying the previous character. The OE indicator is set to a logic 1 upon detection of an overrun condition and reset whenever the CPU reads the contents of the Line Status Register. If the FIFO mode data continues to fill the FIFO beyond the trigger level, an overrun error will occur only after the FIFO is full and the next character has been completely received in the shift register. OE is indicated to the CPU as soon as it happens. The character in the shift register is overwritten, but it is not transferred to the FIFO.

Bit 2: This bit is the Parity Error (PE) indicator. Bit 2 indicates that the received data character does not have the correct even or odd parity, as selected by the even parity

Adresarea registrelor

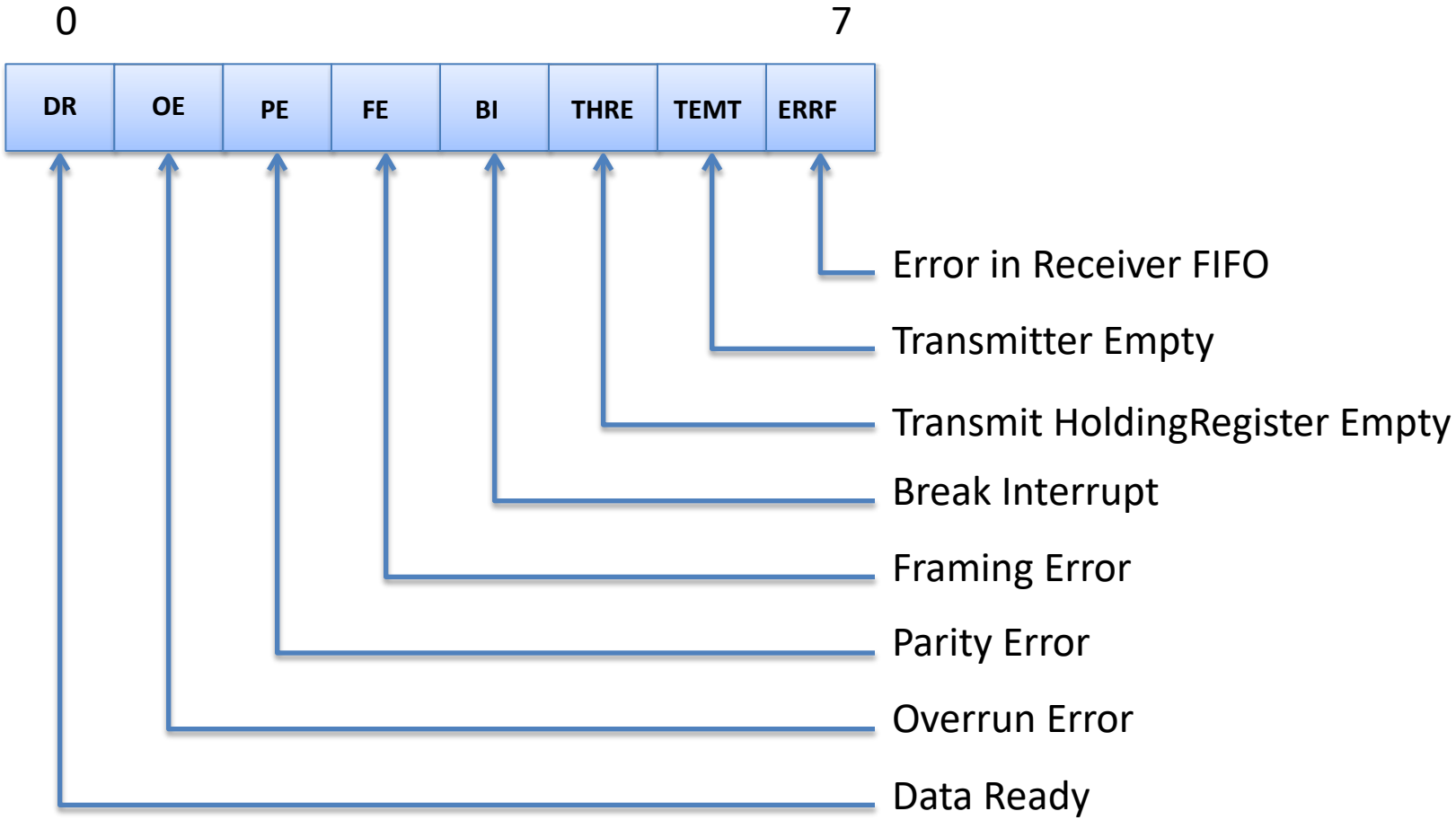
1. Mapate în memorie:
   - Registrele apar ca locații de memorie
   - Accesate folosind load/store (`movb/movw/movl/movq`)
2. "I/O instructions":
   - Spațiu diferit (16 biți) de adrese pentru dispozitive I/O mai vechi
   - Specific (zilele noastre) arhitecturii
   - Instrucțiuni speciale: `inb`, `outb`, etc.
3. Indirecție:
   - Scrie un registru "index" cu un offset, apoi un registru "data" cu valoarea efectivă a registrului
   - Folosit pentru a salva spațiul de adresă (de obicei spațiul I/O)

Registre ns16550 (fiecare are 8 biți)

Addr.	Name	Description	Notes
0	RBR	Receive Buffer Register (read only)	DLAB=0
0	THR	Transmit Holding Register (write only)	DLAB=0
1	IER	Interrupt Enable Register	DLAB=0
2	IIR	Interrupt Identification Register (read only)	
2	FCR	FIFO Control Register (write only)	
3	LCR	Line Control Register	
4	MCR	MODEM Control Register	
5	LSR	Line Status Register	
6	MSR	MODEM Status Register	
7	SCR	Scratch Register	
0	DLL	Divisor Latch (LSB)	DLAB=1
1	DLM	Divisor Latch (MSB)	DLAB=1

DLAB = bit 7 of the LCR register

ns16550 LSR:
Line Status Register



Un driver UART foarte simplu

```
#define UART_BASE 0x3f8
#define UART_THR  (UART_BASE + 0)
#define UART_RBR  (UART_BASE + 0)
#define UART_LSR  (UART_BASE + 5)

void serial_putc(char c)
{
// Wait until FIFO can hold more chars
while( (inb(UART_LSR) & 0x20)== 0);
// Write character to FIFO
outb(UART_THR, c);
}

char serial_getc()
{
// Wait until there is a char to read
while( (inb(UART_LSR) & 0x01) == 0);
// Read from the receive FIFO
return inb(UART_RBR);
}
```

Register addresses from data sheet 0x3f8: location on a PC

Send a character (wait until we can first)

Read a character (spin waiting until one is there to read)

Un driver UART foarte simplu

- De fapt, mult prea simplu!
  - Dar așa arată întotdeauna prima versiune pentru orice…
- Fără cod de inițializare, fără error handling.
- Folosește ***Programmed I/O*** (PIO)
  - CPU citește sau scrie explicit toate valorile din registre
  - Toate datele trec automat prin registrele CPU
- Folosește ***polling***
  - CPU face busy-waiting înainte de fiecare send/receive
    - Buclă strânsă!
  - Nu poate să facă nimic altceva între timp
    - Deși, putem să ne gândim la un sistem de threading și sincronizare…
  - Fără CPU polling, nu putem face nici un fel de operații I/O

Registrele I/O nu sunt memorie

Registrele unui device nu se comportă ca un RAM!

- Conținutul registrelor se schimbă fără scrieri de la CPU
  - Flag-uri (biți) de status
  - Date recepționate
- Scrierile în registre sunt folosite pentru a declanșa acțiuni
  - Trimiterea de date
  - Resetarea unor mașini de stări din periferic

Problema cu cache-urile

- Citirile din I/O nu pot veni din cache
  - Valoarea reg I/O se schimbă ⇒ cache-ul devine **inconsistent**
- Write-back caches (și write buffers) cauzează probleme
  - Nu știi când linia respectivă va fi scrisă efectiv în periferic
- Citirile și scrierile nu pot fi combinate în linii de cache
  - Registrele fac scrierea/citirea la nivel de octet/cuvânt
  - Scrierea unei linii întregi poate să suprascrie și alte registre în afara celor utile
  - Chiar și citirile în masă pot declanșa schimbări în starea unui device

⇒ Accesul la registrele de I/O trebuie să nu implice memoria cache

  - Handling în MMU: PTE-urile au un flag “no cache”
  - Spațiul I/O oricum nu permite caching

Alte probleme

1. Cum evităm să facem polling?
   - Cum știe CPU-ul când device-ul este gata sau a terminat ultima comandă?
2. Cum lăsăm CPU-ul în pace?
   - Putem face transfer de date fără a trece prin CPU sau cache?
   - CPU-ul poate să facă altceva?
3. De unde vin aceste locații de memorie pentru registre?
   - Cum poate SO să găsească dispozitivele mapate în spațiul fizic de adresă?
   - Cum sunt alocate adresele fizice?

Evităm cu totul polling-ul

- Un CPU nu poate să facă poll pentru fiecare dispozitiv ca să vadă dacă este gata
  - Pierdere de timp
  - Durează prea mult ca să reacționeze
- Soluția: dispozitivele pot să *întrerupă* procesorul
  - Funcționează ca o excepție: salvează starea curentă și sare la o adresă prestabilită din memoria alocată kernelului
  - Informația legată de sursa întreruperii este codificată în *vector*

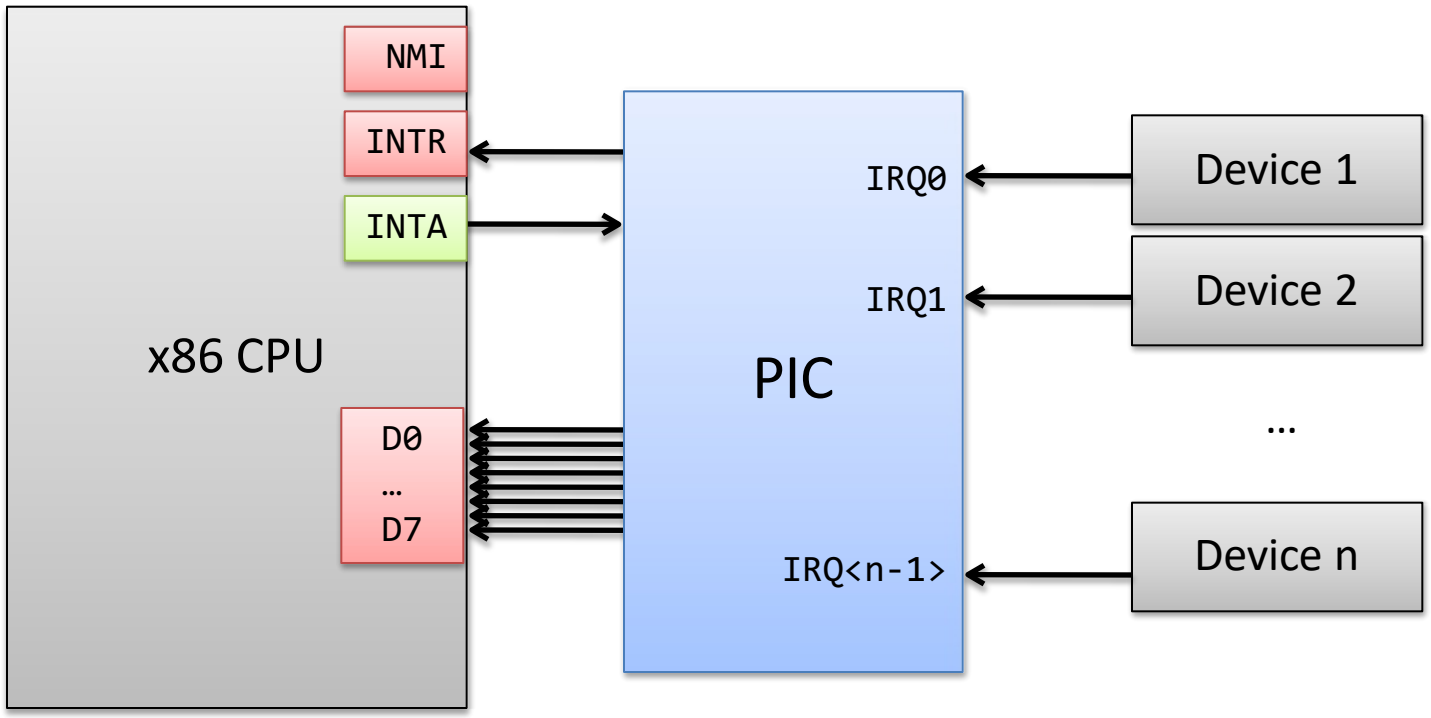
Întreruperi

- CPU are linii **Interrupt-request** declanșate de dispozitivul I/O
  - Pot fi declanșate pe nivel logic sau pe front
- **Interrupt handler** recepționează cererea de întrerupere
- **Mascabile,** pentru a evita sau întârzia anumit întreruperi
- Vectorul de întrerupere face corespondența între întrerupere și handler-ul corect
  - Bazate pe priorități
  - Unele **nonmascabile**

Vectori excepție la x86

0	Divide error
1	Debug exception
2	**Non-maskable interrupt (NMI)**
3	Breakpoint
4	Overflow
5	Bounds check
6	Invalid opcode
7	Coprocessor not available
8	Double fault
9	Coprocessor segment overrun (386 or earlier)
A	Invalid task state segment
B	Segment not present
C	Stack fault
D	General protection fault
E	Page Fault
F	Reserved
10	Math fault
11	Alignment check
12	Machine check
13	SIMD floating point exception
14-1F	Reserved to Intel
20-FF	**Available for external interrupts**

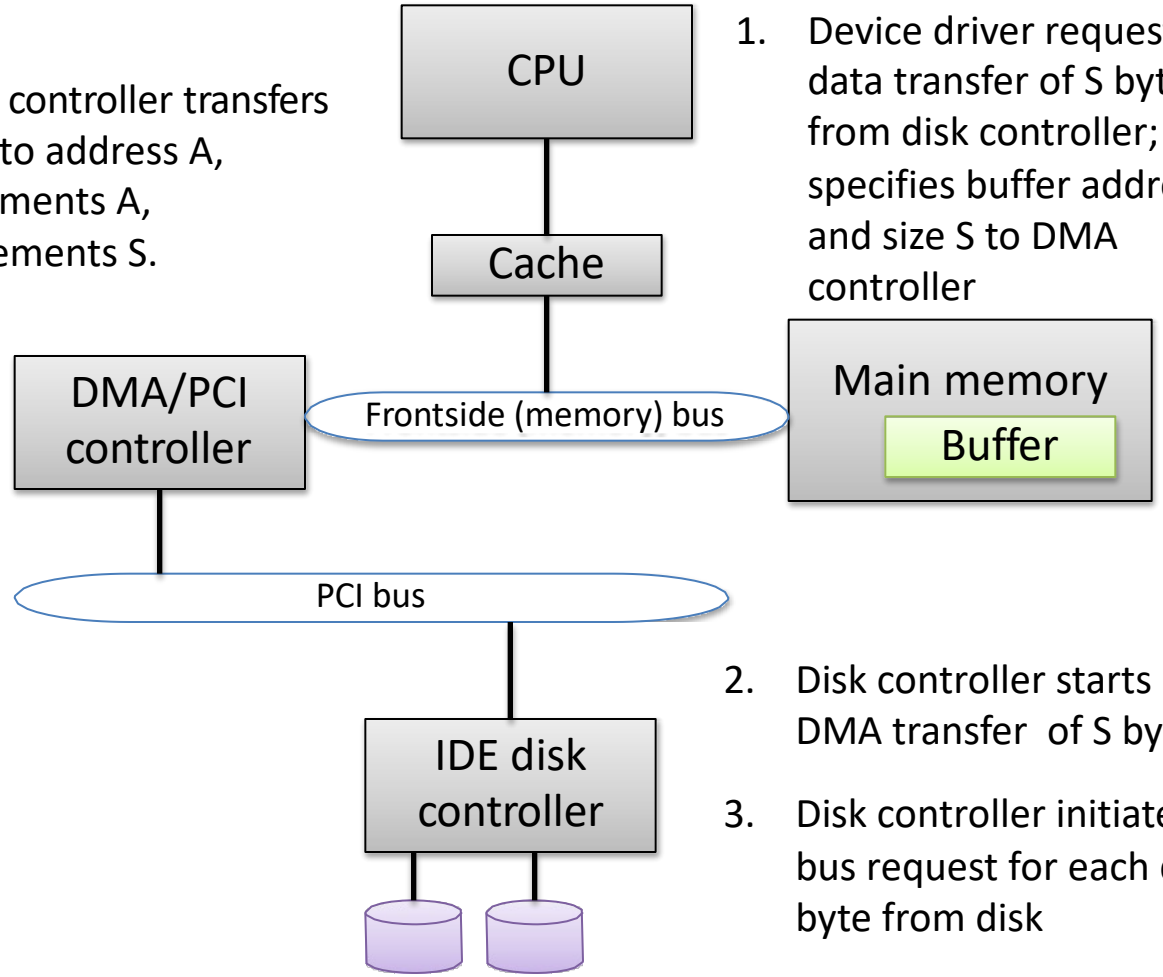
Programmable Interrupt Controller (PIC)



Direct Memory Access

- Evită *I/O programat* pentru majoritatea datelor
  - De ex. rețea rapidă sau interfețe de disc
- Necesită un *controller DMA*
  - De obicei implementat în procesor, în zilele noastre
- Ocolește CPU și transferă datele direct de la I/O la memorie
  - Nu blochează CPU-ul
  - Salvează lățimea de bandă
  - Doar o singură întrerupere pe transfer

Transfer DMA old-style



1. Device driver requests data transfer of S bytes from disk controller; specifies buffer address A and size S to DMA controller

2. Disk controller starts DMA transfer of S bytes

3. Disk controller initiates bus request for each data byte from disk

4. DMA controller transfers byte to address A, increments A, decrements S.

5. When S == 0, DMA controller interrupts CPU to indicate transfer complete

Avantajul de bază al DMA

- **Decuplează** transferul de date de procesarea de date
  - CPU nu trebuie să copieze datele de la/la dispozitiv
  - Nu poluează cache-ul CPU
  - Pot fi procesate când decide CPU (sau SO)
  - Performanță mărită: CPU și device I/O merg în paralel
- Dezavantaje posibile:
  - Overhead mare pentru transferuri foarte mici de date
  - De obicei nu e o problemă (până și UART-urile fac DMA!)

DMA și memoria Cache

- DMA înseamnă ca memoria devine **inconsistentă** cu cache-ul CPU
- Opțiuni:

1. CPU poate să marcheze bufferele DMA ca non-cacheable

⇒ large hit – probabil vrea să proceseze datele oricum

2. Cache poate să facă “snoop” la tranzacțiile DMA (dar nu scalează foarte bine la sistemele multiprocesor)
3. SO poate să golească/invalideze explicit regiuni din cache

⇒ cache management este o parte importantă a driverelor de dispozitive!

DMA și Memoria Virtuală

- Adresele DMA sunt **fizice**
  - Apar pe magistrala externă
- Codul utilizatorilor și al SO lucrează cu adrese **virtuale** (în majoritatea timpului)
- SO (și device drivers) trebuie să translateze manual virtual ↔ fizic atunci când programează controllerele DMA
  - Acest lucru necesită mai mult de o tablelă hardware de pagini!
  - DMA al unei singure regiuni de adrese virtuale s-ar putea să **nu aibă un corespondent contiguu** în spațiul de adrese fizice
  - **Scatter-gather** DMA controllers: DMA de la/la o listă de regiuni
- Sisteme foarte recente: implementează IOMMU
  - Funcționează ca MMU, dar pentru DMA scrie din dispozitive
  - Tot trebuie programat de SO pentru a fi în concordanță cu starea MMU

Unde sunt toate aceste registre?

- De unde știe SO câte dispozitive I/O sunt conectate?
- Unde sunt mapate registrele dispozitivelor în spațiul fizic?
  - Și căror vectori de întrerupere le corespund?
- Soluție: include-le în designul *magistralei de I/O*
  - Exemplu: PCI

PCI este…

Peripheral **C**omponent **I**nterconnect

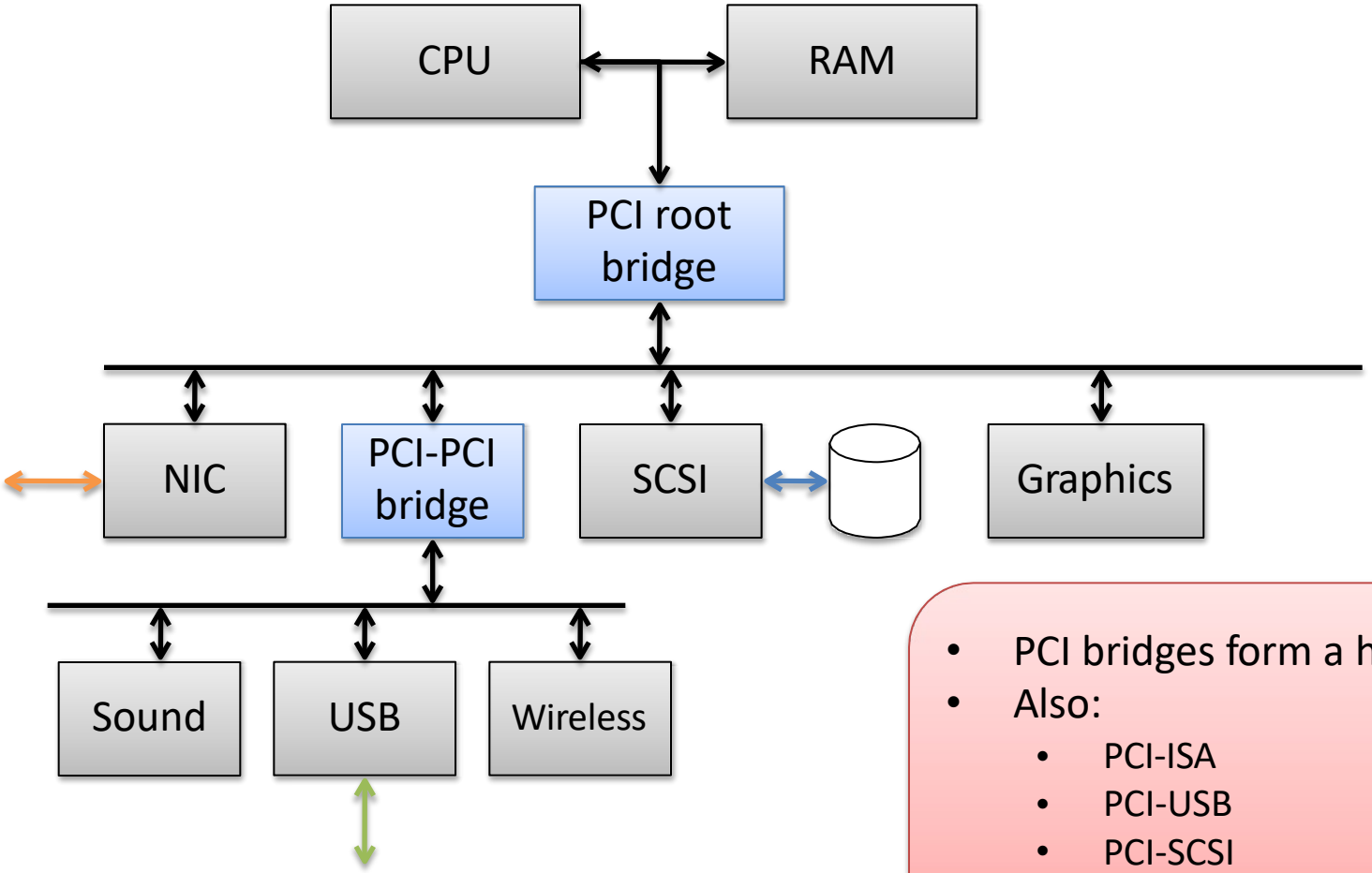
- Standard electric pentru conectarea dispozitivelor
  - Ca și PCMCIA, PCI-X, PCI-Express, etc.
- Un standard pentru conectorii fizici
- Un set de ”protocoale de bus” pentru comunicația dintre dispozitive
- O interfață vizibilă software-ului pentru operații I/O hardware

PCIe a urmat PCI, dar extinde aceeași interfață software

PCI încearcă să rezolve probleme multiple:

- Device discovery
  - Detecția tuturor dispozitivelor din sistem
- Alocarea de adrese
  - La ce adrese apar diferitele registre ale dispozitivelor conectate?
- Interrupt routing
  - Ce semnale de întrerupere de la fiecare dispozitiv se mapează și în ce vector de excepție?
- DMA inteligent
  - Dispozitive ce au “Bus mastering” nu mai au nevoie de controller DMA

Conexiuni fizice: PCI este un arbore



- PCI bridges form a hierarchy
- Also:
  - PCI-ISA
  - PCI-USB
  - PCI-SCSI
  - Etc.

Spațiul de adresă PCI este plat

- Fiecare device PCI cere un set de adrese
  - Spațiu fizic de adrese (32-biți sau 64-biți)
  - Spațiu adresă I/O (de obicei 16-biți)
- Rezultat:
  - Fiecare dispozitiv apare ca un segment contiguu de adrese
    - În spațiul de memorie
    - În spațiul I/O (doar pe x86)

Dispozitivele PCI sunt self-describing

- Fiecare dispozitiv are un header pentru configurație

  - Accesat prin bridge-ul părinte, inițial

- Câteva câmpuri:

Bits	Description
16	Manufacturer ID (idenfies Intel, 3Com, NVidia, etc.
16	Model ID (specific to manufacturer)
24	Class code (what kind of device is this?)
8	Version identifier

- Plusuri:
  - Alocă automat spațiul de adresă
  - Intreruperi
  - Informații de natură electrică
  - Etc.

Localizarea tuturor dispozitivelor

- Găsește bridge-ul PCI “root”
  - PCI bridge este în vârful arborelui
  - PC-urile mari pot avea mai mult de unul
- Citește configurațiile pentru a găsi toate dispozitivele atașate
  - Adaugă la lista de dispozitive și funcții
  - Înregistează cerințele pentru spațiul de adresă
  - If a bridge, recurse!
- Rezultatul:
  - Lista completă a tuturor dispozitivelor din sistem cu toate cerințele de spațiu de adresă aferente

Alocarea adreselor

- Găsește adresele pentru fiecare dispozitiv și bridge Cerințele includ:
  - Fiecare dispozitiv are dată dimensiunea spațiului de adresă necesar
  - Toate dispozitivele de “sub” un bridge au adrese care sunt incluse în spațiul de adresă al bridge-ului
  - Fiecare bridge are un segment de memorie care include toate segmentele de memorie ale tuturor “copiilor”.
  - Fiecare segment este limitat de adrese putere a lui 2
- Apoi programează:
  - Fiecare bridge PCI cu informații legate de translatarea adreselor
  - Fiecare dispozitiv cu registre “base-address/range” (BAR)

Întreruperi PCI

- Patru linii de întrerupere
  - INTA, INTB, INTC, INTD…
  - Bridge-urile permit cablarea arbitrară a liniilor de IRQ a dispozitivelor la cele patru linii
  - Translatate de root bridge în intreruperi de sistem
- PCI Express introduce MSI
  - Message-signalled interrupts
  - Întreruperea codificată ca un PCI write într-un spațiu anume de adrese
  - Translatate de root bridge în intreruperi de sistem
  - Întreruperile pot fi rutate individual către un anumit core/APIC

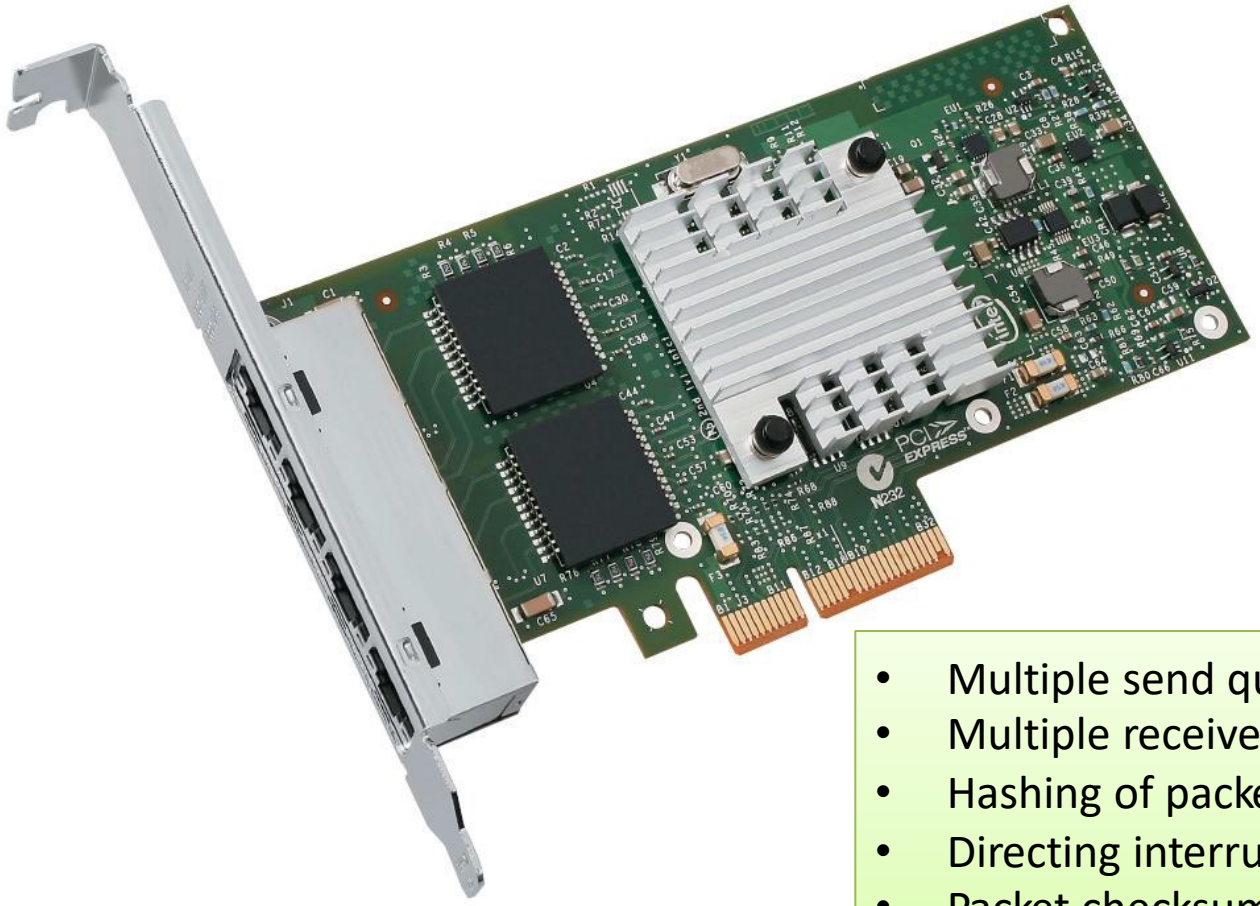
DMA peste PCI

- PCI permite **Bus Mastering**
  - Device-ul poate emite tranzacții de scriere/citire oriunde în memorie
  - Chiar (în anumite cazuri) spre alte dispozitive PCI
- Controller-ul DMA extern nu mai e relevant
  - Controller integrat în dispozitivul însuși
  - Principiul se aplică: device-ul face DMA pentru date de la/la memorie
  - Mult mai flexibil / dispozitive inteligente

Dispozitive inteligente

- Bus mastering, plus foarte mult spațiu de adresă acum
- Dispozitivele pot acum să acceseze autonom orice:
  - Locație din memoria principală
  - Alte dispozitive
- Permite protocoale complexe pentru interacțiunea CPU ↔ Device
  - Încearcă să țină și CPU și device-ul ocupat în perioadele de activitate mare
  - In RAM buffering
  - “Descriptor rings” – mecanism de schimbare de cereri și răspunsuri

Exemplu: Intel e1000 PCI- Express Ethernet card



- Multiple send queues
- Multiple receive queues
- Hashing of packet headers to queues
- Directing interrupts to different cores
- Packet checksumming in hardware
- etc.

Rezumat

- Dispozitivele și CPU comunică via:
  - Registre I/O mapate în memorie
  - Întreruperi și vectori de întrerupere
  - Direct Memory Access (DMA)

- Magistralele I/O (precum PCI):
  - Permit dispozitivelor să partajeze/aloce adrese fizice
  - Alocă întreruperi
  - Permit bus mastering direct memory access

Acknowledgements

- These slides contain material developed and copyright by:
  - Arvind (MIT)
  - Krste Asanovic (MIT/UCB)
  - Joel Emer (Intel/MIT)
  - James Hoe (CMU)
  - John Kubiatowicz (UCB)
  - David Patterson (UCB)

- MIT material derived from course 6.823
- UCB material derived from course CS252