

Fundamentos de Arquitetura de Computadores

Tiago Alves

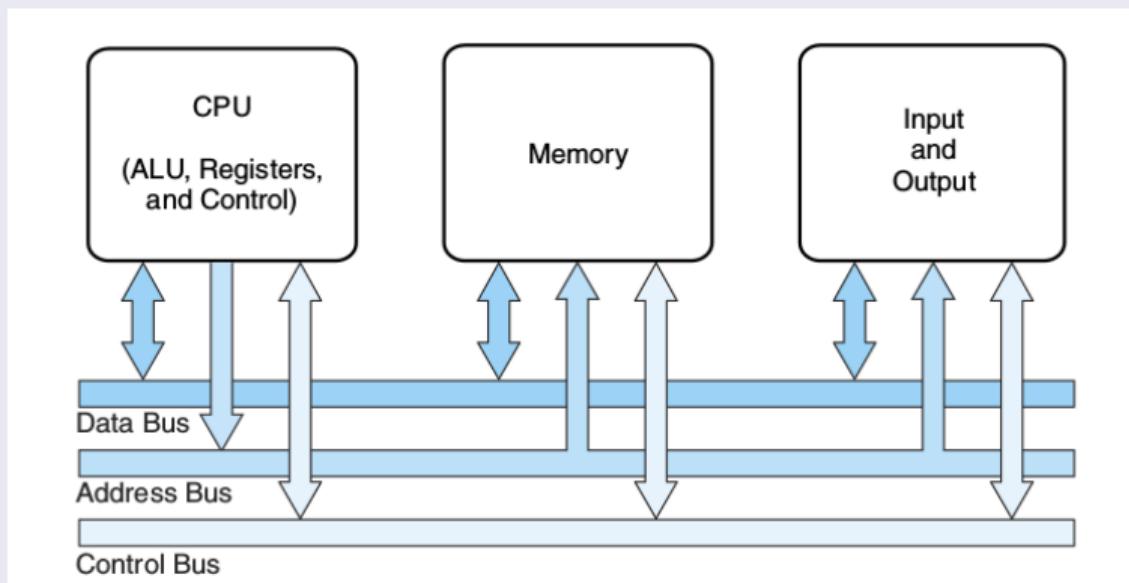
Faculdade UnB Gama
Universidade de Brasília



Definição

Ligações elétricas (ou caminhos) compartilhadas por múltiplos dispositivos.

Costumam ser as vias por onde trafegam dados, sinais de controle e instruções que são trocados entre os diferentes módulos funcionais de um computador digital.

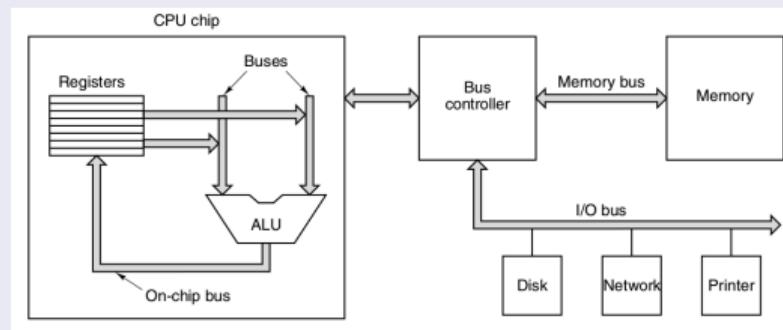


Definição

Onde encontrá-los?

- internos à CPU, usados na troca de dados com a ULA.
- externos à CPU, usados para ligá-la à memória (barramento do sistema) ou a dispositivos de I/O (portas).

A interligação do computador a um dispositivo externo ao sistema computacional depende de barramentos diferenciados. Costumamos chamar esses barramentos de **portas**.



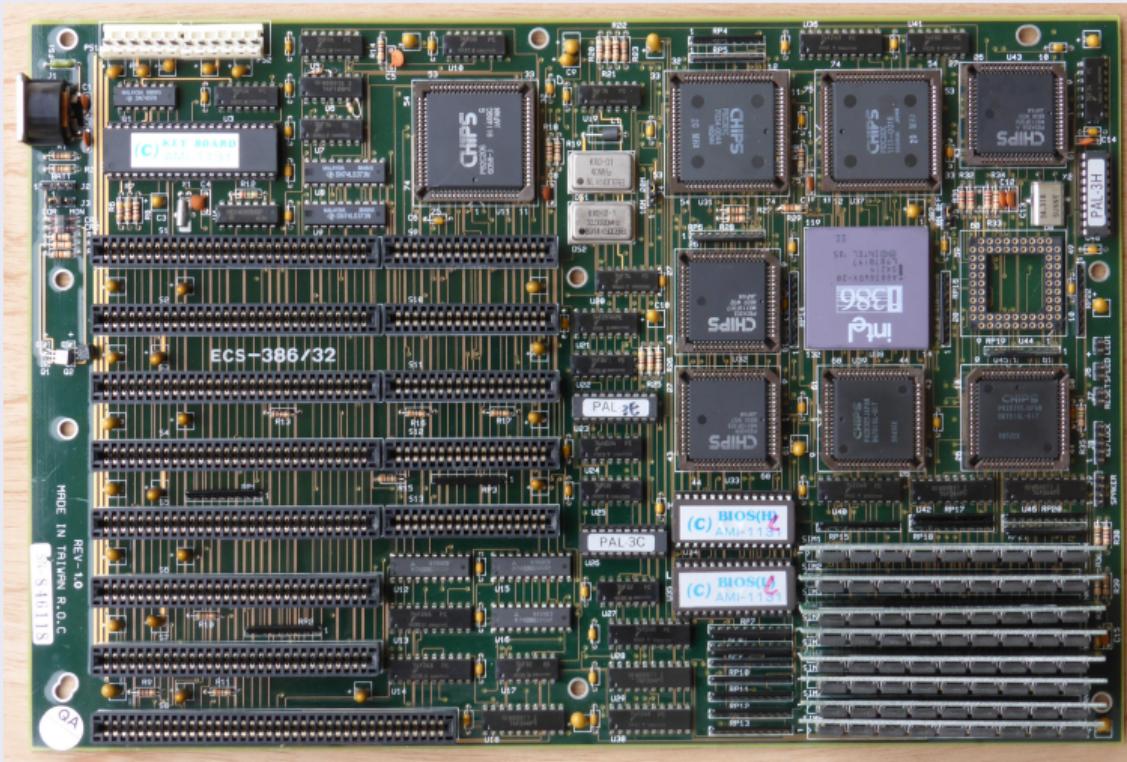
Primórdios da Estrutura em Sistemas Eletrônicos Computacionais

- Computadores pessoais primitivos possuíam apenas 1 barramento externo
- 50 a 100 linhas paralelas de cobre repousadas na placa-mãe
- Conectores eram espaçados a distâncias regulares. Eram substrato para a conexão de memórias e de placas/cartões de I/O
- Computadores modernos possuem barramento de propósito especial entre a CPU e a memória principal e outros barramentos para I/O.

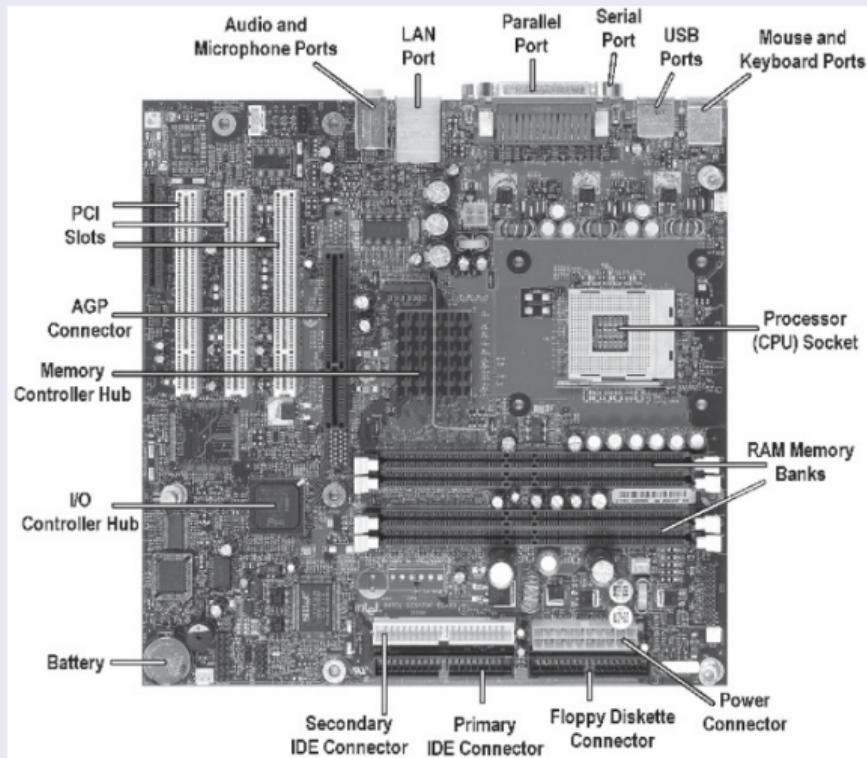


Barrantos

Interfaceamento



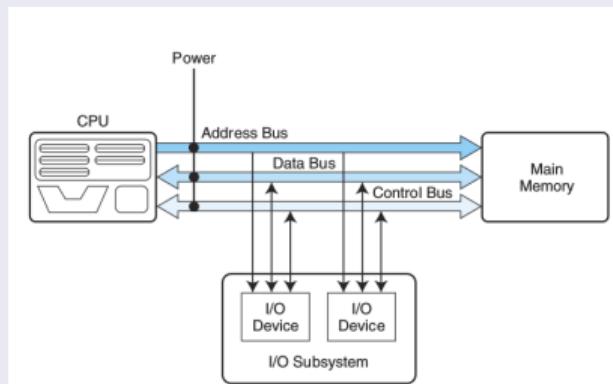
Interfaceamento



Esquema lógico básico de um Barramento

Lembra uma CPU: possui endereços, dados e linhas de controle.

- potência: provê a potência (energia) necessária para o funcionamento dos componentes interligados ao barramento;
- endereços: carrega a informação sobre a localização (na memória, por exemplo) a partir da qual dados deverão ser lidos ou escritos;
- controle: indica qual dispositivo tem autorização para usar o barramento e com qual propósito;
- dados: usadas para movimentação de dados.



Esquema lógico básico de um Barramento

Esquema simplificado para 16 bits.



Esquema lógico básico de um Barramento

Transação típica através de barramento:

- envio de endereço (para leitura ou escrita);
- transferência de dados da memória para um registrador (leitura) ou de um registrador para a memória (escrita).

Essa mesma dinâmica é seguida caso se deseje E/S (I/O) com periféricos.

Essas transferências costumam respeitar esquema de temporização.



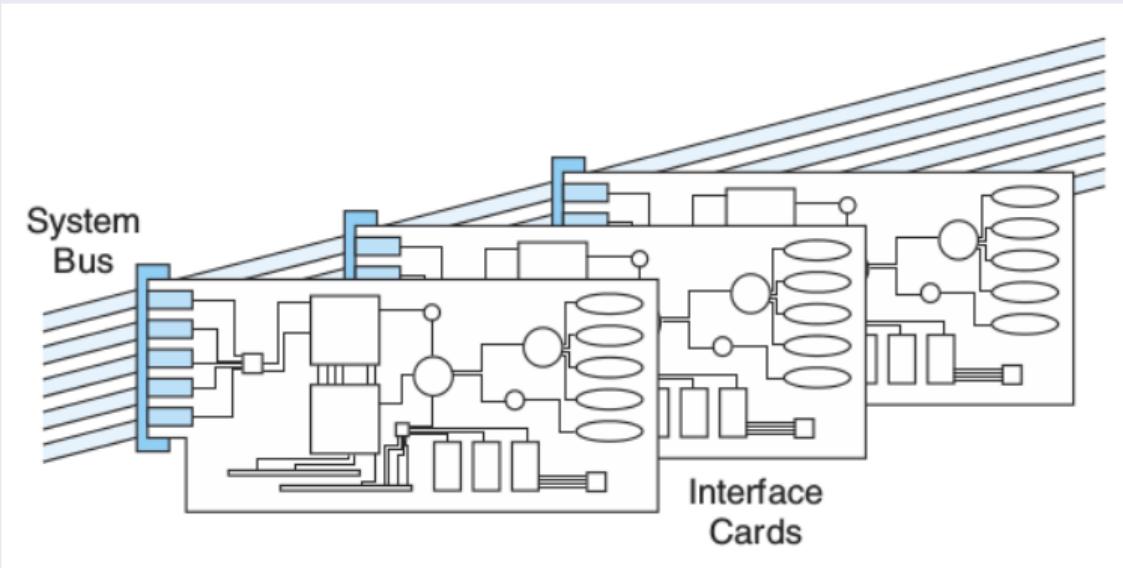
Barramentos em Sistemas Computacionais

Tipos:

- processador-memória (barramento interno): curtos, altíssima velocidade, ajustado ao sistema de memória para entregar a maior largura de banda possível nas transações entre CPU e a memória. Bastante específico a determinada arquitetura (de produção).
- E/S (I/O): mais longos que o anterior e permite a interação entre diferentes tipos de dispositivos com largura de bandas diferentes. Projeto mais genérico com ênfase em compatibilidade.
- barramento *backplane*: geralmente construído no chassis da máquina e conecta a CPU, os dispositivos de I/O e a memória. É um barramento compartilhado.



Barramentos em Sistemas Computacionais

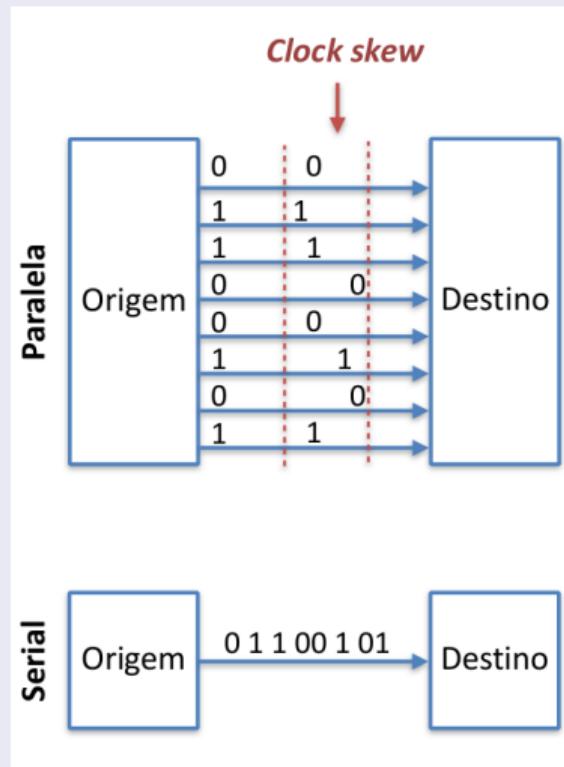


Barramentos em Sistemas Computacionais

Tipos: (quanto à forma como os dados são enviados)

- serial: possui um conjunto reduzido de linhas físicas (minimamente 1 linha) através do qual os dados a serem transferidos são encaminhados 1 bit por vez.
- paralelo: possui várias linhas dispostas paralelamente de forma que é possível encaminhar mais de 1 bit de dados por vez. Em geral, conduzem 1 byte ou 1 palavra por vez.





Operação de Barramentos

Agentes

- ativos ou mestres
- passivos ou escravos

Exemplos

- troca de dados entre CPU e disco
- CPU (mestre) ordena o controlador de disco (escravo) a leitura ou escrita de um bloco
- na resposta à requisição, os papéis poderão mudar: controlador (mestre) envia à memória principal as words lidas do disco.

Intermediário elétrico: **bus drivers, receivers e transceivers**. Reforço dado ao dispositivo na interação com o barramento.



Operação de Barramentos

| Master | Slave | Example |
|-------------|-------------|--|
| CPU | Memory | Fetching instructions and data |
| CPU | I/O device | Initiating data transfer |
| CPU | Coprocessor | CPU handing instruction off to coprocessor |
| I/O device | Memory | DMA (Direct Memory Access) |
| Coprocessor | CPU | Coprocessor fetching operands from CPU |



Modelo Básico de um Barramento

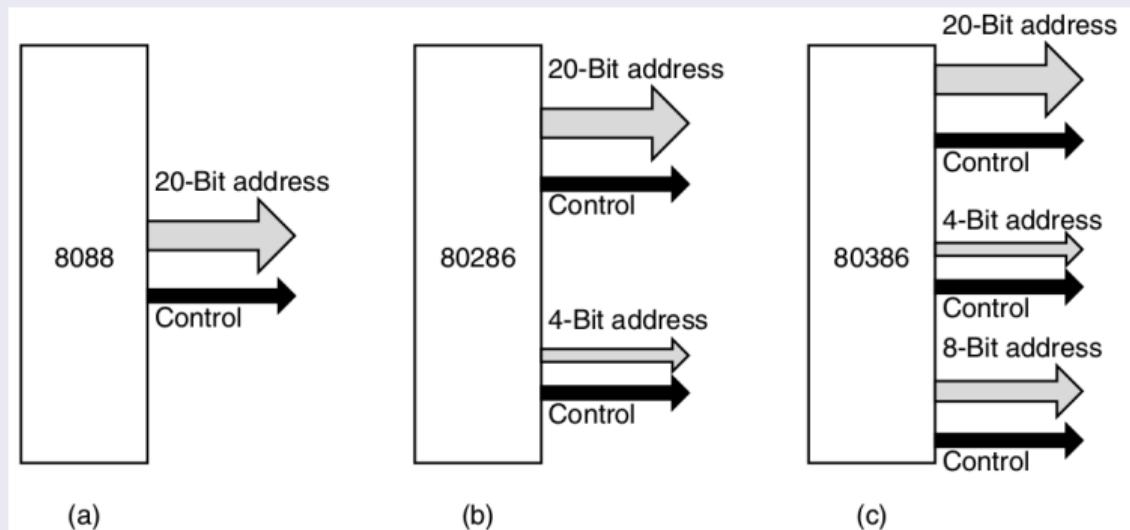
Largura de Barramento:

- Regra básica: quanto mais linhas de endereço possuir um barramento, mais memória poderá ser endereçada pela CPU
- Mais linhas de endereços implica mais linhas elétricas. Que demandam mais espaço físico e que, ao final, resultam em conectores grandes. Isso encarece projetos eletrônicos!



Largura de Barramento

Projeto da Intel de linhas de endereço e suas idiossincrasias: retrocompatibilidade vs. aumento de desempenho.



Largura de Barramento

8086

e.g. Intel 8086 from 1970's
(pre-MIPS)

https://en.wikipedia.org/wiki/Intel_8086



40 pins in total. 16 were "AD".
This was a 16 bit processor.

| | | MAX MODE | { MIN MODE } |
|------|----|----------|---|
| GND | 1 | 40 | V _{CC} |
| AD14 | 2 | 39 | AD15 |
| AD13 | 3 | 38 | A16/S3 |
| AD12 | 4 | 37 | A17/S4 |
| AD11 | 5 | 36 | A18/S5 |
| AD10 | 6 | 35 | A19/S6 |
| AD9 | 7 | 34 | BHE/S7 |
| AD8 | 8 | 33 | MN/MX |
| AD7 | 9 | 32 | R _O |
| AD6 | 10 | CPU | R _O /GTO (HOLD) |
| AD5 | 11 | 30 | R _O /GT _I (HOLDA) |
| AD4 | 12 | 29 | LOCK (WR) |
| AD3 | 13 | 28 | S ₂ (M/I/O) |
| AD2 | 14 | 27 | S ₁ (DT/R) |
| AD1 | 15 | 26 | S ₀ (DEN) |
| ADO | 16 | 25 | OS ₀ (ALE) |
| NMI | 17 | 24 | OS ₁ (INTA) |
| INTR | 18 | 23 | TEST |
| CLK | 19 | 22 | READY |
| GND | 20 | 21 | RESET |

Largura de Barramento

O aumento de linha de dados tende a acompanhar o aumento no número de linhas de endereços

Busca por desempenho: aumento da largura de banda do barramento

- Estratégia 1: diminuir o período de relógio do barramento (mais transferências por segundo)
- Estratégia 2: aumentar a largura do barramento de dados (mais bits por transferência).

Essa busca esbarra em problemas de retrocompatibilidade e de compatibilidade eletromagnética

Solução de compromisso em algumas tecnologias: barramento multiplexado. Ora trafegam endereços, ora trafegam dados.



Sincronização de Barramento

Categorias de barramento:

- Síncronos: provê linha de sinal de relógio. Oscilador a cristal.

Todas as interações com o barramento devem respeitar a frequência de relógio.

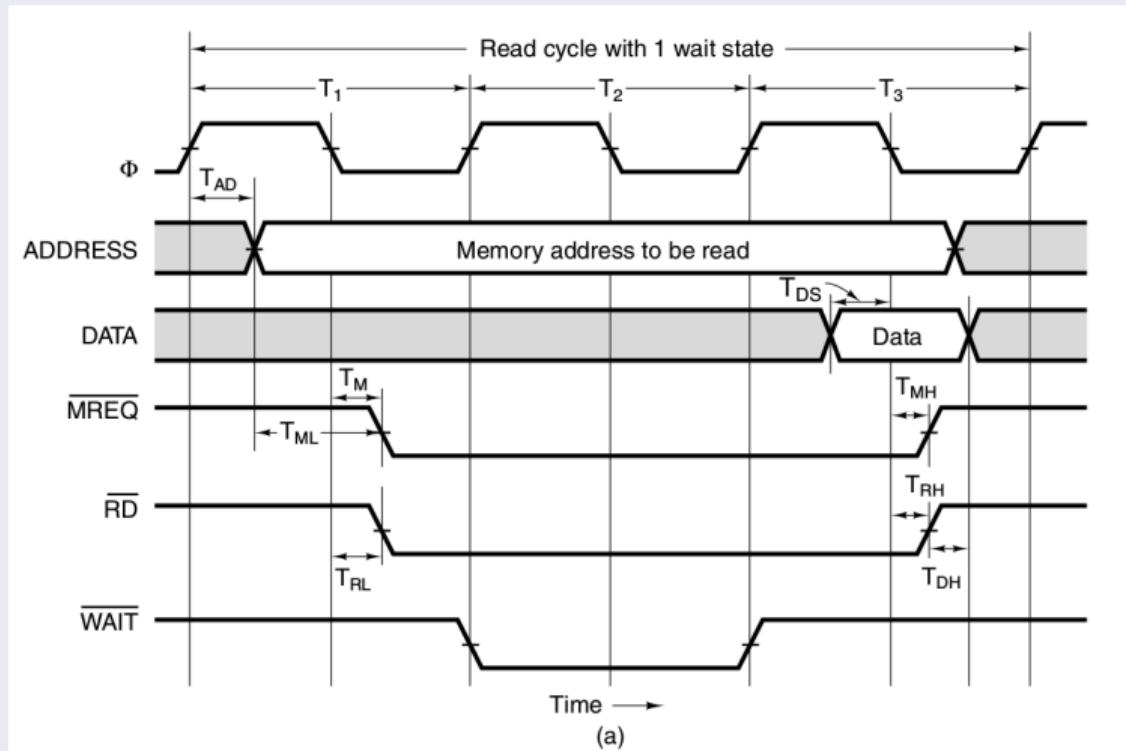
- Assíncronos: não possuem um sinal de relógio mestre/de referência.

Sinais de relógio poderão ser de qualquer comprimento e não precisam ser os mesmos entre todos os pares de dispositivos



Sincronização de Barramento

Síncrono (operação de leitura):



Sincronização de Barramento

Síncrono:

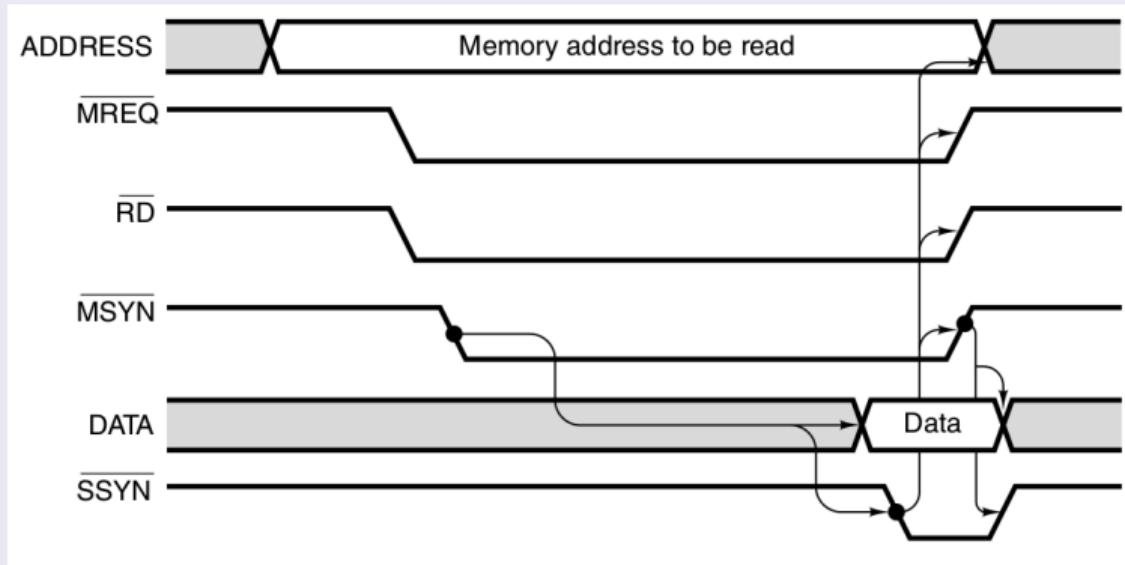
| Symbol | Parameter | Min | Max | Unit |
|----------|--|-----|-----|------|
| T_{AD} | Address output delay | | 4 | nsec |
| T_{ML} | Address stable prior to \overline{MREQ} | 2 | | nsec |
| T_M | \overline{MREQ} delay from falling edge of Φ in T_1 | | 3 | nsec |
| T_{RL} | RD delay from falling edge of Φ in T_1 | | 3 | nsec |
| T_{DS} | Data setup time prior to falling edge of Φ | 2 | | nsec |
| T_{MH} | \overline{MREQ} delay from falling edge of Φ in T_3 | | 3 | nsec |
| T_{RH} | \overline{RD} delay from falling edge of Φ in T_3 | | 3 | nsec |
| T_{DH} | Data hold time from negation of \overline{RD} | 0 | | nsec |

(b)



Sincronização de Barramento

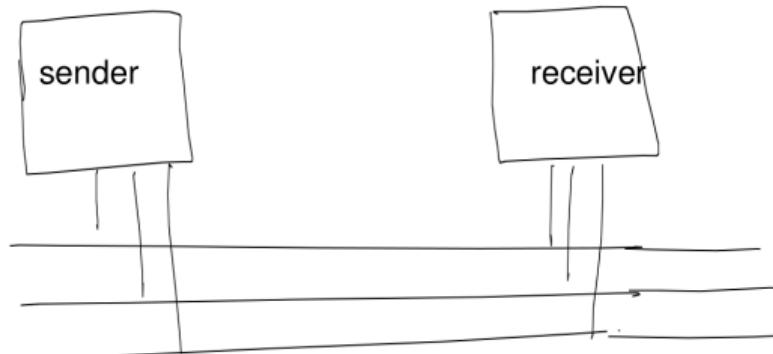
Assíncrono:



Sincronização de Barramento: Assíncrono

Handshaking:

"Handshaking" : one method for an asynchronous bus



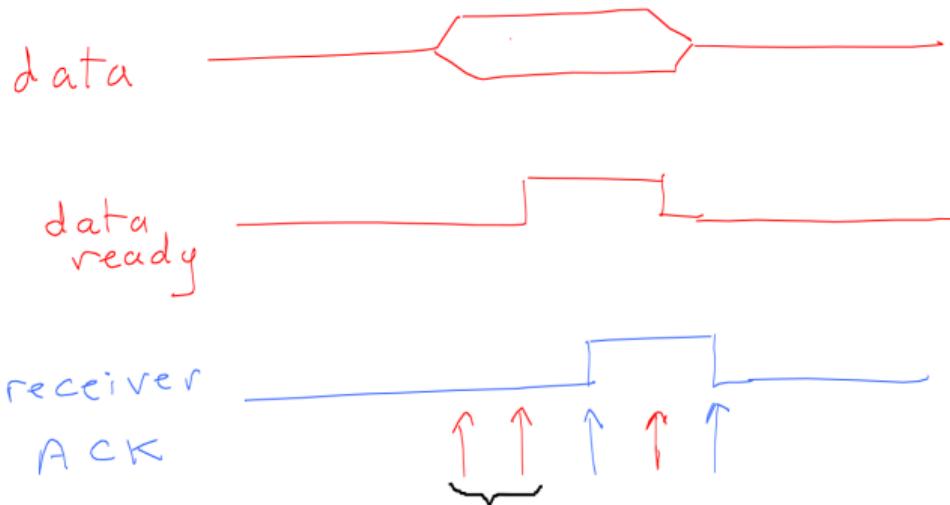
It can be initiated by source (sender) or by destination (receiver).

Sincronização de Barramento: Assíncrono

Iniciado pelo emissor:

sender initiated

e.g. printer controller to printer



delay of 'data ready' is needed here because of variability of signal on (physical) wire

Sincronização de Barramento: Assíncrono

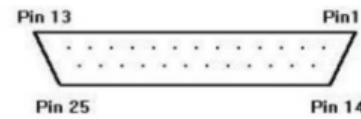
Porta Paralela:



Example:

"parallel port" for (pre-USB)
printer

25 Pin Parallel Port [female]:



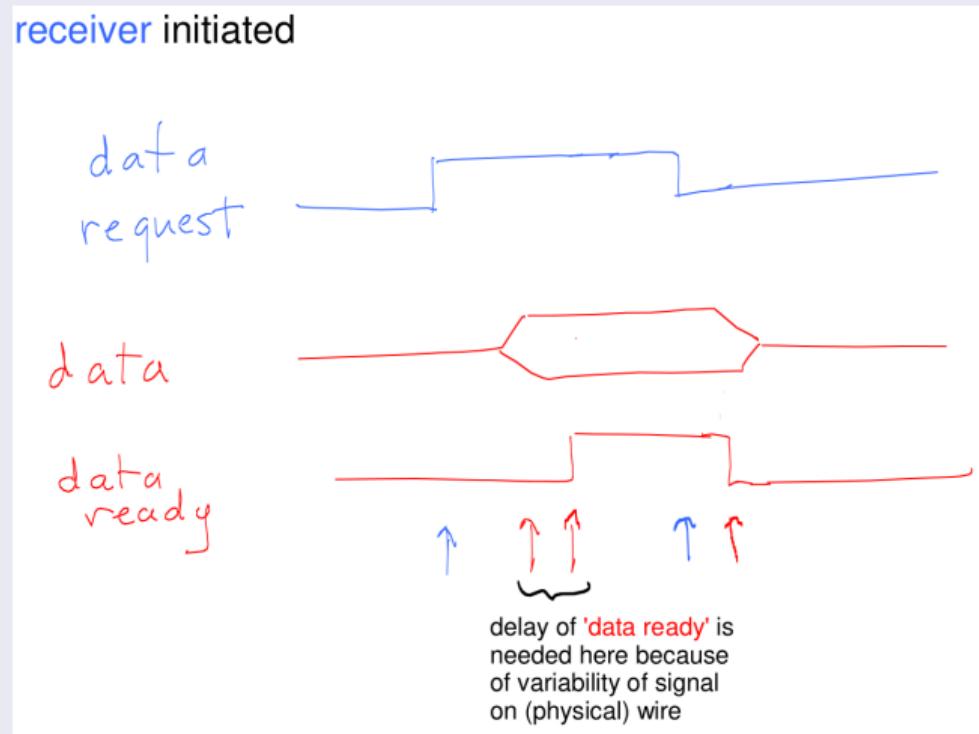
| Pin | Description | Pin | Description |
|-----|--------------|-----|-------------|
| 1 | /STROBE | 14 | /AUTOFEED |
| 2 | DATA 0 | 15 | /ERROR |
| 3 | DATA 1 | 16 | /INIT |
| 4 | DATA 2 | 17 | /SELECT |
| 5 | DATA 3 | 18 | GROUND |
| 6 | DATA 4 | 19 | GROUND |
| 7 | DATA 5 | 20 | GROUND |
| 8 | DATA 6 | 21 | GROUND |
| 9 | DATA 7 | 22 | GROUND |
| 10 | /ACKNOWLEDGE | 23 | GROUND |
| 11 | BUSY | 24 | GROUND |
| 12 | PAPER-END | 25 | GROUND |
| 13 | SELECTED | | |

data ready

receiver ACK

Sincronização de Barramento: Assíncrono

Iniciado pelo receptor:



Sincronização de Barramento: Assíncrono

Iniciado pelo emissor (barramento compartilhado):

e.g. **sender initiated**

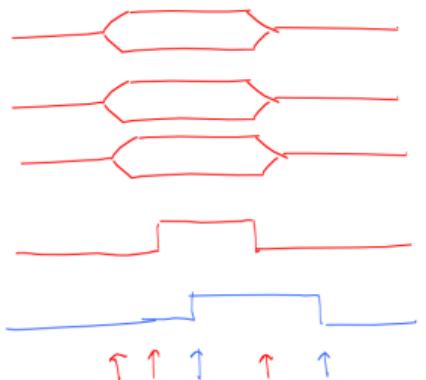
address (receiver)

data

control

data ready

receiver ACK

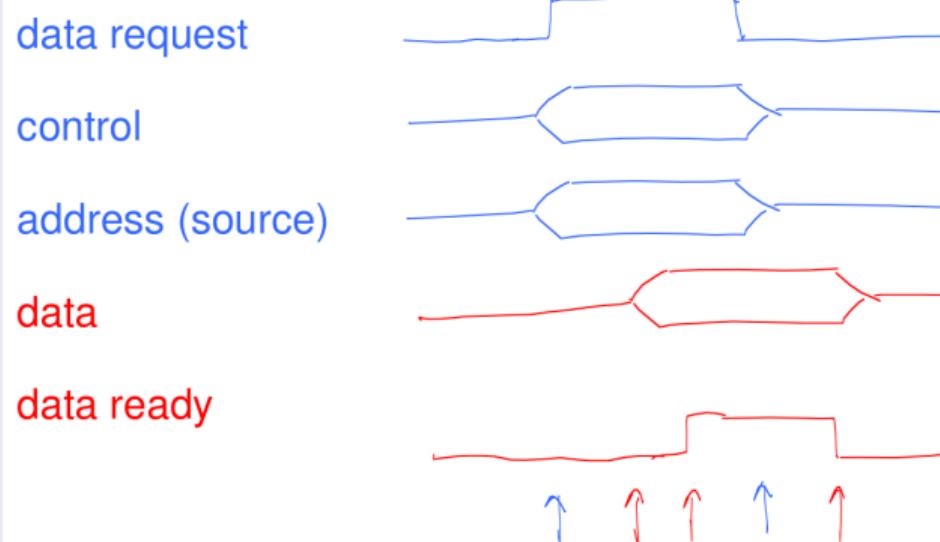


Note that address, data, and control may be many bits each.

Sincronização de Barramento: Assíncrono

Iniciado pelo receptor (barramento compartilhado):

e.g. receiver initiated



Controle (Arbitragem) de Barramento

O modelo em que há um mestre e um escravo é fácil de se analisar e de se implementar

Porém, a arquitetura mais frequente permite com que haja troca de papéis entre dispositivos a depender das transações:

- chips de IO necessitam se tornar mestres para lerem e escreverem na memória principal e para lançar interrupções
- coprocessadores (FPU) podem também precisar se tornarem mestres

Dois esquemas para controle de barramento no cenário em que mais de um dispositivo necessitar se tornar mestre:

- centralizado
- descentralizado



Controle (Arbitragem) de Barramento: Centralizado

Há um árbitro/juiz que decide quem controlará o barramento em determinado momento

Embutido em CPU ou, às vezes, acompanhante da CPU

Linha única de pedido de controle de barramento usada para sinalizar o pedido para a CPU (OR)

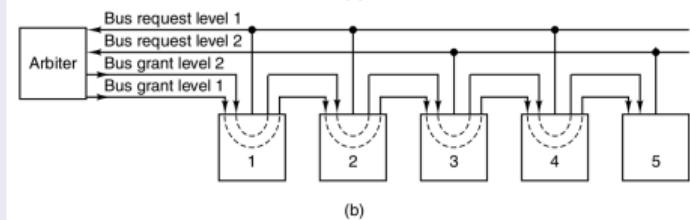
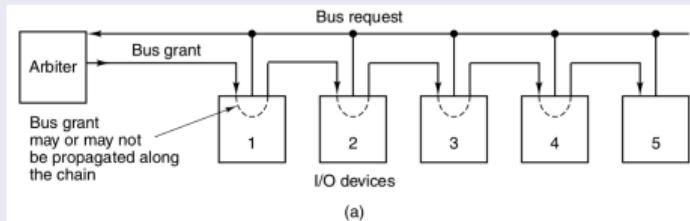
- múltiplos pedidos podem se sobrepor!
- duas opções/estados: sem pedidos ou com pedidos.



Controle (Arbitragem) de Barramento: Centralizado

Árbitro conectado aos dispositivos para os quais cederá controle (do barramento) através de uma linha serial interconectada

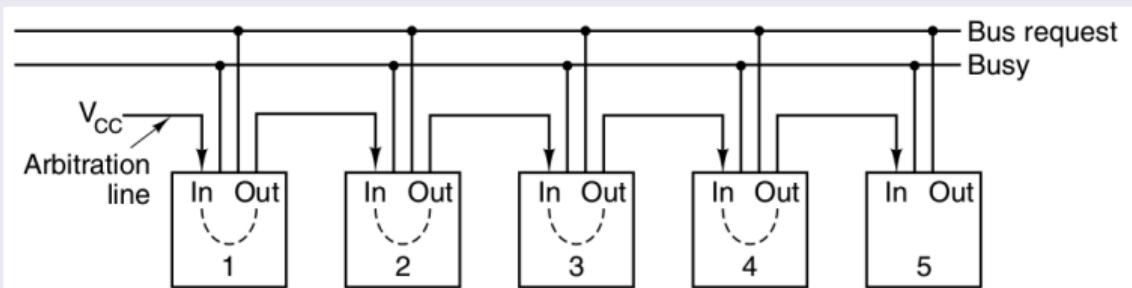
- O dispositivo mais próximo cuja demanda foi enviada recebe o sinal e controla o barramento.
- Se o dispositivo que recebeu o sinal não demandou controle, deve encaminhar o sinal de autorização pela linha serial para seu próximo vizinho.
- É possível alterar o esquema de interconexão para melhorar a distribuição de prioridades entre os dispositivos que compartilham o barramento.



Controle (Arbitragem) de Barramento: Descentralizado

Demanda aumento do número de linhas e uma sistemática de sensibilização dos dispositivos que compartilham o barramento

- Os dispositivos deverão monitorar as linhas e eventuais pedidos, controlando o barramento apenas quando julgarem ser a sua vez.
- Impõe um limite na razão entre o número de dispositivos e o número de linhas de requisição



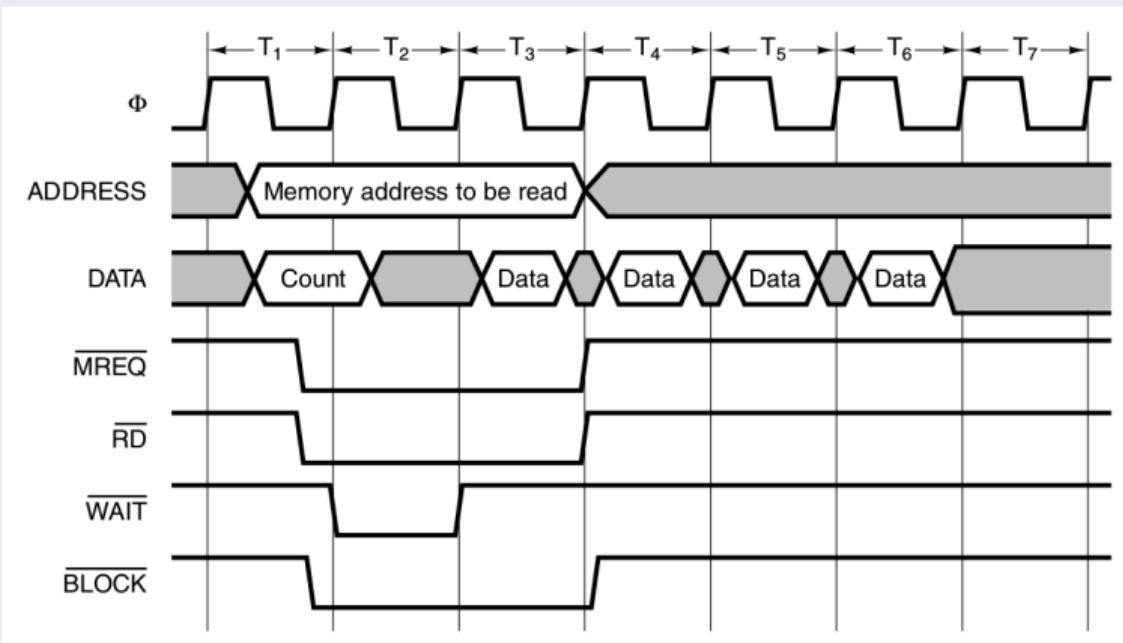
Operações em Barramento

Módulos construtivos que compõem sistemas eletrônicos trocam palavras através de barramentos.

- Quando esquemas de caching são empregados, as trocas costumam ser em blocos de palavras: aumenta a eficiência das transferências diluindo a latência do tempo de aquisição de controle do barramento.
- O mestre costuma indicar quantas palavras irá transmitir pelo barramento.

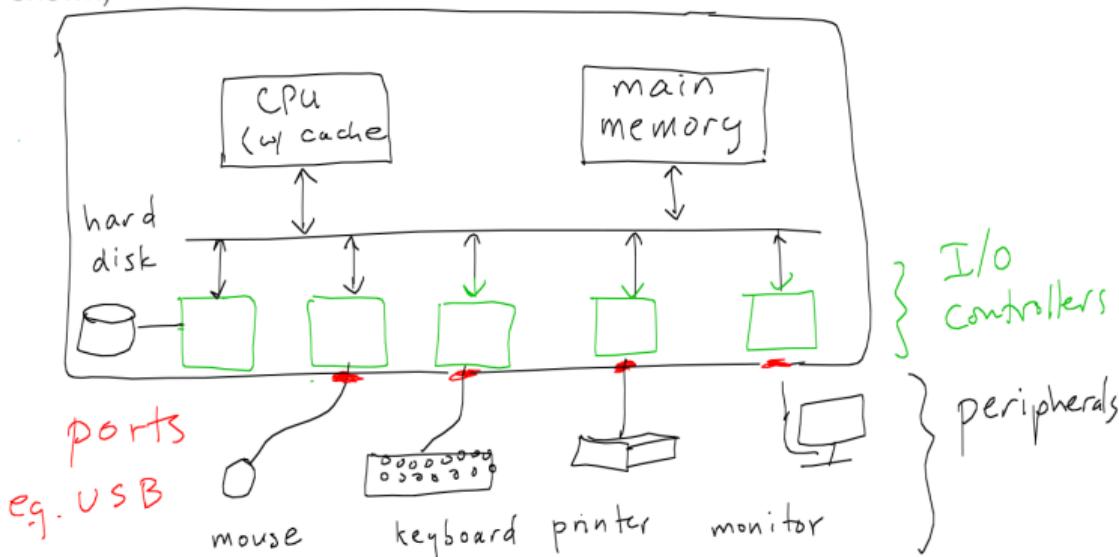


Operações em Barramento

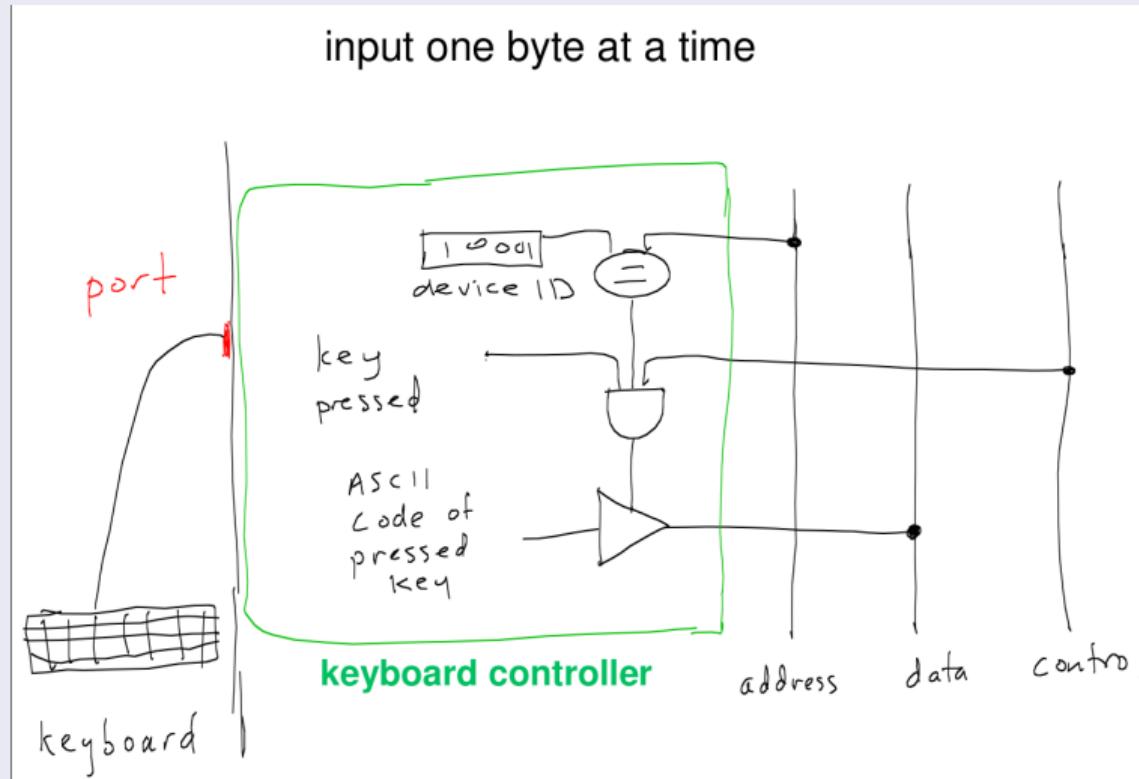


Periféricos e controladores de I/O (Input-Output)

- interface between devices and system bus
- contain registers, clock, memory, processor ("driver software" is part of OS and thus is in main memory or on hard disk -- not shown)



Periféricos e controladores de I/O: Teclado



Periféricos e controladores de I/O: Mouse



mouse contains
sensors (button,
XY position, ...)

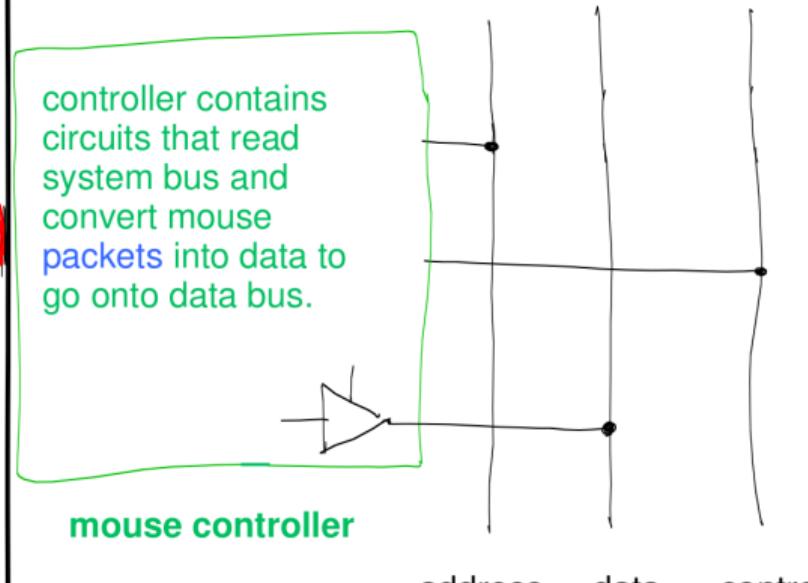
Packets of data
are sent along
wire:

- button pressed?
- X, Y position
- scroll

e.g. 2: Mouse (*historical*)

controller contains
circuits that read
system bus and
convert mouse
packets into data to
go onto data bus.

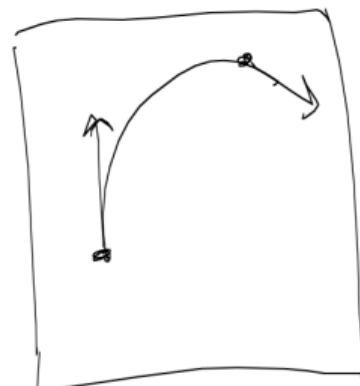
mouse controller



address data control

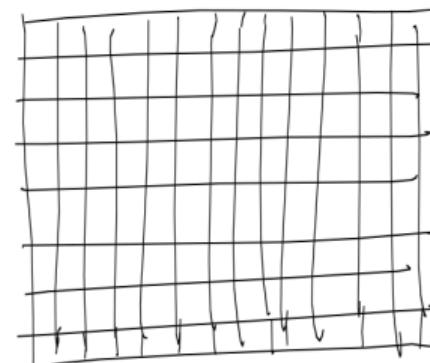
e.g. 3 output device: screen

“vector graphics”



draw commands
(contour position,
direction, thickness)

“raster graphics”



pixels
(picture elements)

Computer Graphics and Video

suppose your computer is generating the images e.g. game

$$| 1000 \times 1000 \times 3 \times 30 \text{ fps}$$

rows

columns

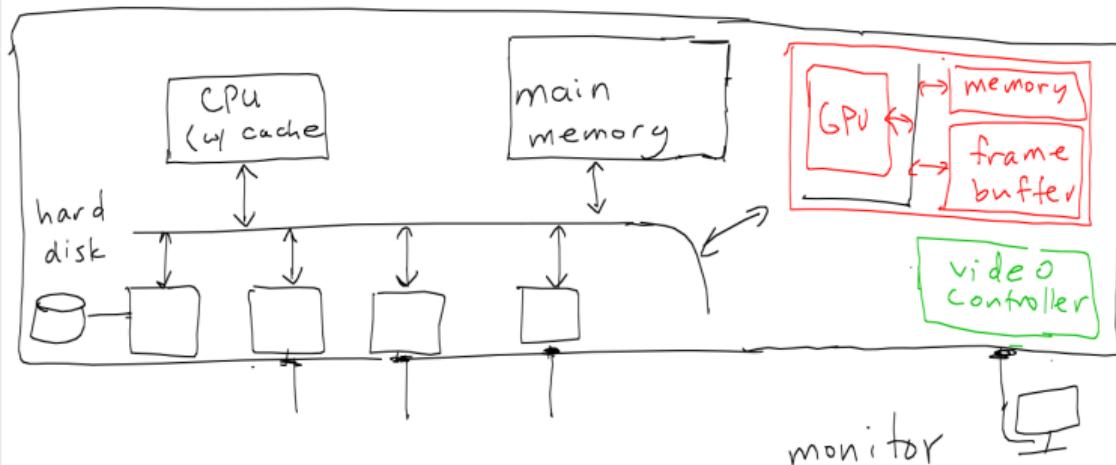
RGB

image
frames /sec

$$\approx 100 \text{ MB/sec}$$

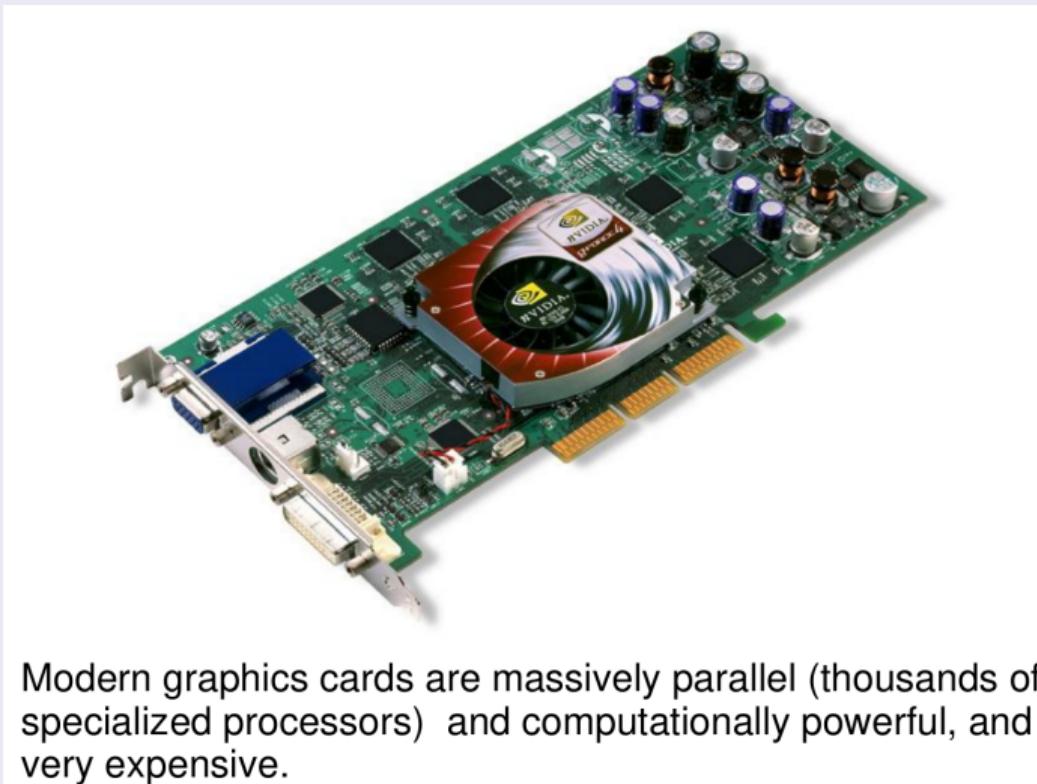
Making images is a lot of work, and sending images to display on the system bus would create a bottleneck.

Graphics cards (industry grew quickly in 1990s)



Instead, the CPU sends 'high level' instructions to the GPU (graphics card) and the GPU does the work of making and displaying the images.

Periféricos e controladores de I/O: Video



Modern graphics cards are massively parallel (thousands of specialized processors) and computationally powerful, and very expensive.

Esquemas de I/O: MMIO

"Memory mapped" I/O (MMIO) is a different approach.

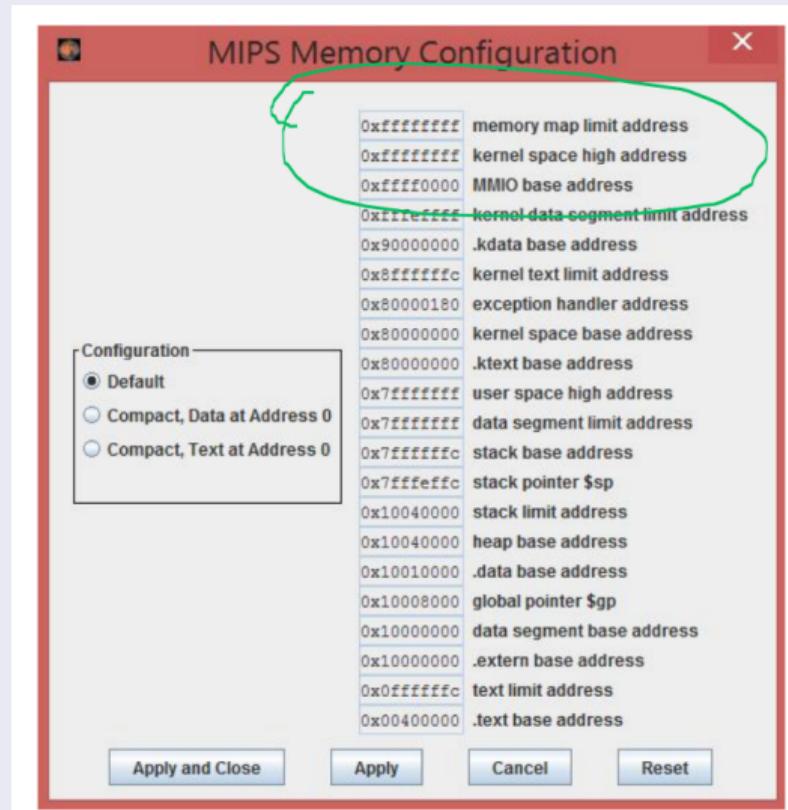
MMIO is used in real MIPS processors.

It uses addresses from 0xffff0000 to 0xffffffff (in kernel).
(64 KB ~ 2^{16} bytes)

MARS can be configured to use MMIO.



Esquemas de I/O: MMIO



Esquemas de I/O: MMIO

console output data register



0x fffff000c

console output control register

0x fffff0008



LSB is ready bit

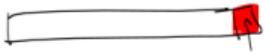
console input data register

0x fffff0004



console input control register

0x fffff0000



LSB is ready bit

Esquemas de I/O: MMIO

```
lui      $t0, 0xffff  
lw       $s0, 4($t0)
```

It only loads a byte, but let's ignore that detail.

console input data register

0x fffff0004



What happens physically ?

There is a circuit that detects that this is a memory mapped address, and that loads the word from the register instead....

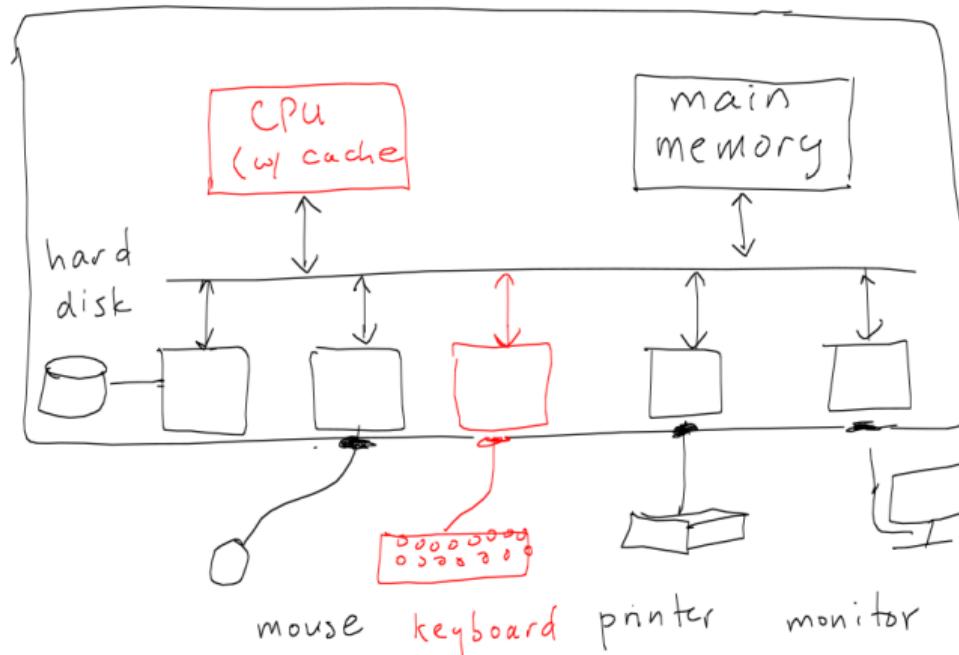
Esquemas de I/O: MMIO



lw \$s0, \$t0

Rather than reading from memory in the usual way, there would be a branch to the I/O exception handler.

Let's examine this in more detail (AND MORE GENERAL).
(console input = keyboard)



Esquemas de I/O: MMIO

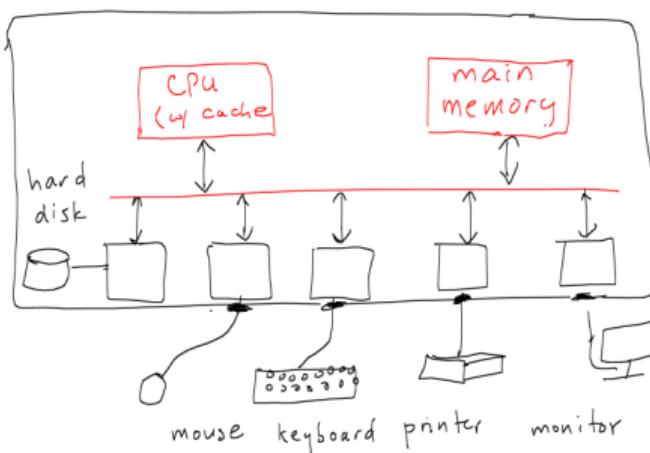
- branch to kernel's I/O exception handler
- CPU translates virtual address 0xffff0004 to I/O device address, namely keyboard ID (details not specified. hardware implementation dependent)
- CPU write I/O device address on the address bus and sets ReadIO control to 1
- keyboard controller puts byte onto the data bus (last lecture)
- CPU reads data bus
- kernel jumps back to user program, which continues execution (and byte is loaded into register)



Esquemas de I/O: MMIO

Note similarity to TLB or instruction/data cache miss.

Addresses, data, controls need to be put on bus.



Esquemas de I/O: MMIO

- same idea for I/O output

lui \$t0, 0xfffff

sw \$s0, 12(\$t0)



Esquemas de I/O: Polling

We sort of skipped an issue:

CPU to device : "Ready or not ? "

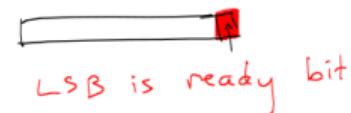
- Does input device have a (new) input ?
- Has output device displayed the previous output (s) ?



Program controlled I/O: Polling (Are you ready? Are you ready?)

console input control register

0x ffff 0000



LSB is ready bit

console input data register

0x ffff 0004



lui \$t0, 0xffff

Wait:

```
lw    $t1, 0($t0)      # load from the input control register
andi $t1, $t1, 0x0001  # reset (clear) all bits except LSB
beq $t1, $zero, Wait  # if not ready, then loop back
```

```
lw    $s2, 4( $t0)      # input device is ready, so read
```

Esquemas de I/O: Polling

Escrita:

Polling (e.g. output)

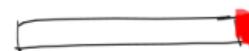
console output data register



0x fffff000C

console output control register

0x fffff0008



LSB is ready bit

lui \$t0, 0xffff

Wait:

lw \$t1, 8(\$t0)
andi \$t1, \$t1, 0x0001
beq \$t1, \$zero, Wait

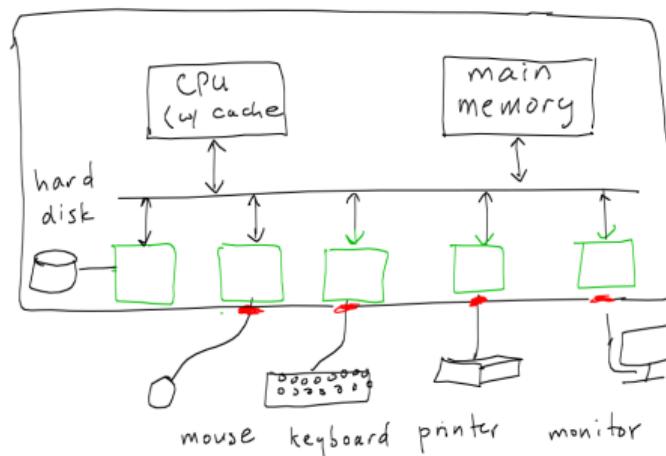
load the output control reg
reset all bits except LSB
if not ready, then loop back

sw \$s2, 12(\$t0)

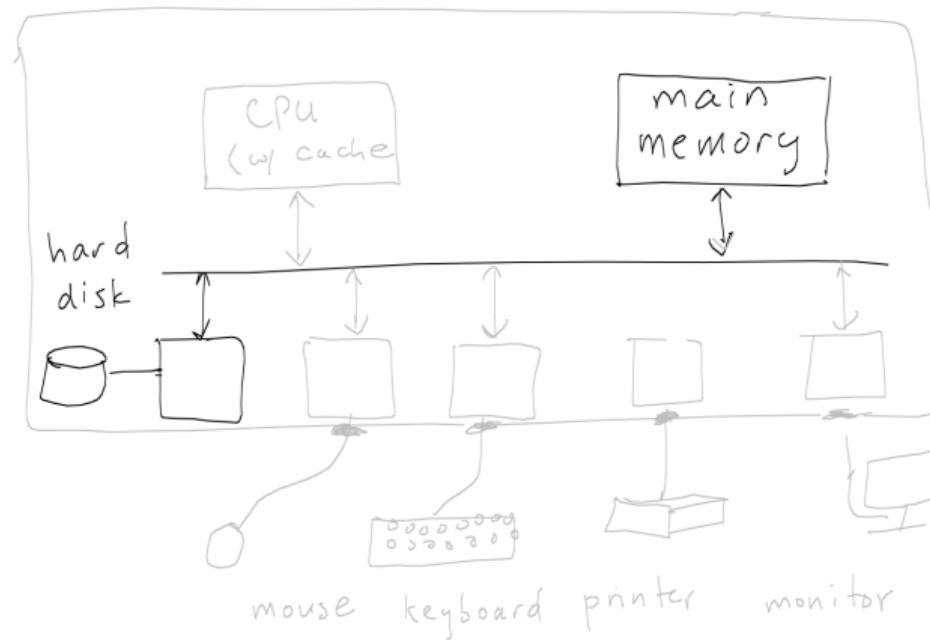
output device ready, so write

Esquemas de I/O: DMA

Suppose page fault occurs. Then, kernel must coordinate a page swap. How ? (sketch)



Direct Memory Access (DMA)

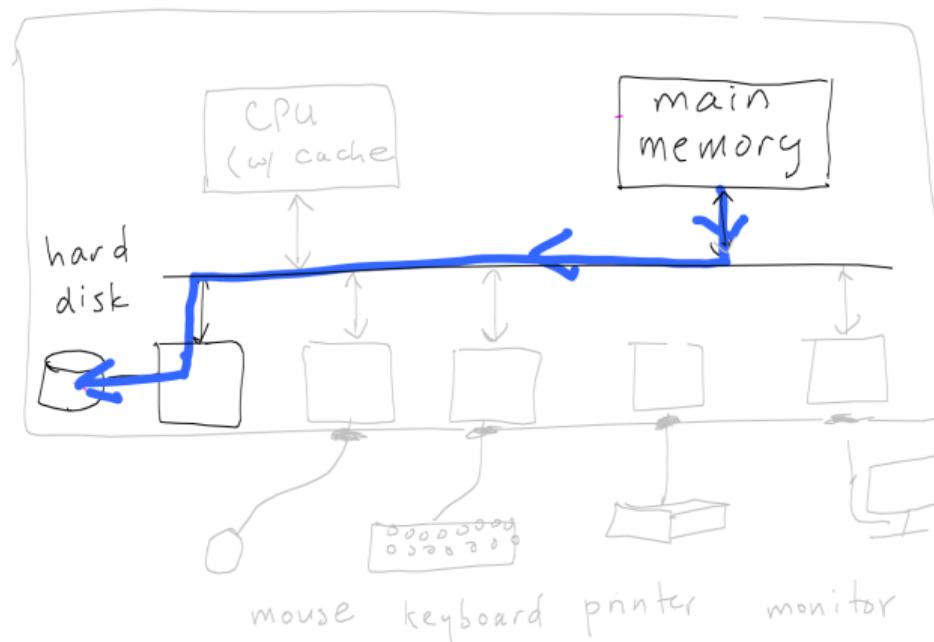


DMA (for page fault): Step 1

CPU writes onto system bus

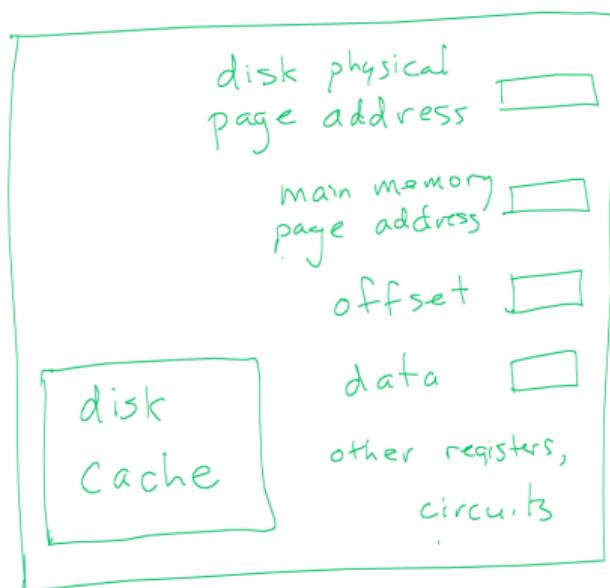
- HDD controller's address (device ID)
(on address bus)
- instruction for HDD controller
(on control bus)
- physical *address* of pages
*(on *data* bus)*

DMA (for page fault): Step 2 (desired)



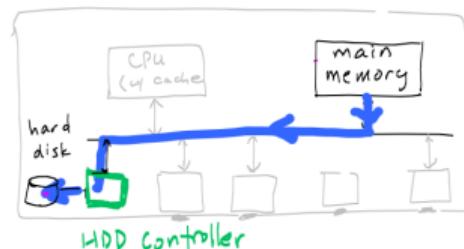
Esquemas de I/O: DMA

HDD controller takes over.

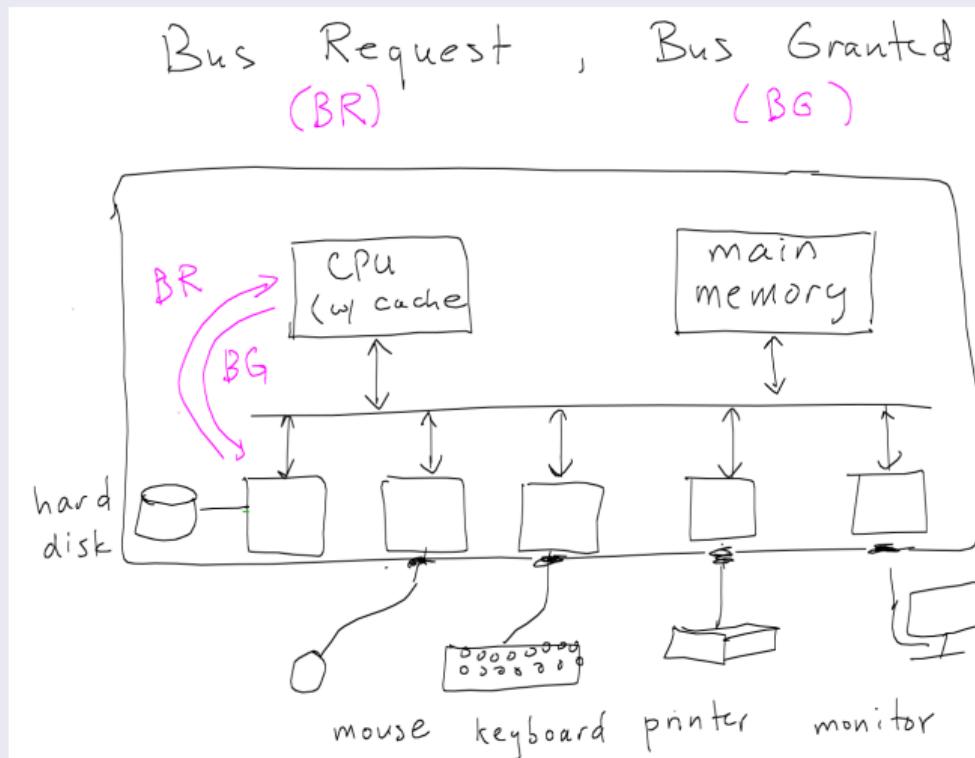


HDD controller

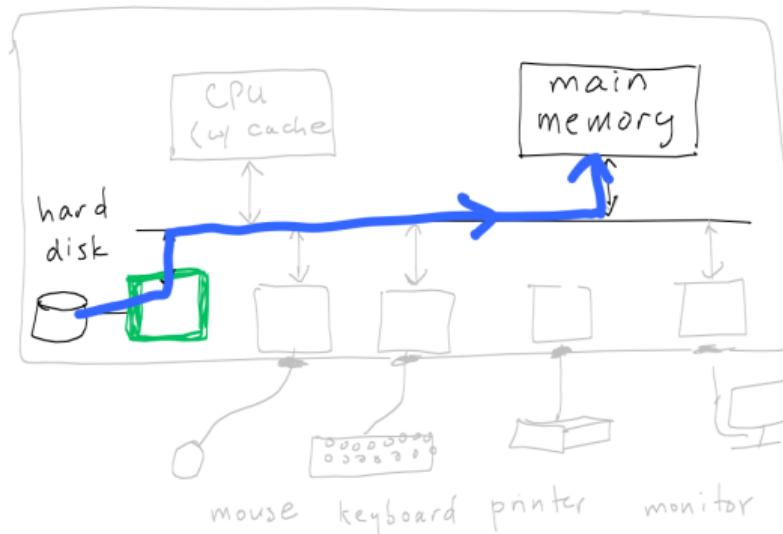
- gets **write** access to system bus (**How?**)
- instructs main memory to write on the system bus, and reads the page word by word, storing in local memory ("disk cache" on previous slide) on the HDD controller (**How ?**)
- transfers the page from disk cache to the hard disk when the HD is at the right physical position



Esquemas de I/O: DMA



Similar steps for transferring page from HDD to main memory.



Esquemas de I/O: Interrupções

(External) Interrupts

I/O device to CPU: "stop what you are doing and do Y"

e.g. keyboard (echo/render to display, d<enter> causes action)
mouse (render to display)
printer (out of paper, render message to user)

- interrupt request (IRQ) lines are used to make requests
(not system bus)

Similar general idea as with DMA.
But there are important differences.



Esquemas de I/O: Interrupções

General questions about interrupt requests

- How does an I/O device make an interrupt request ?
- How to coordinate multiple I/O devices that may *all* make interrupt requests ? Can one device interrupt another ?
- What does the CPU do when it gets an interrupt request ?

Let's examine the first two questions. I'll look at three methods.

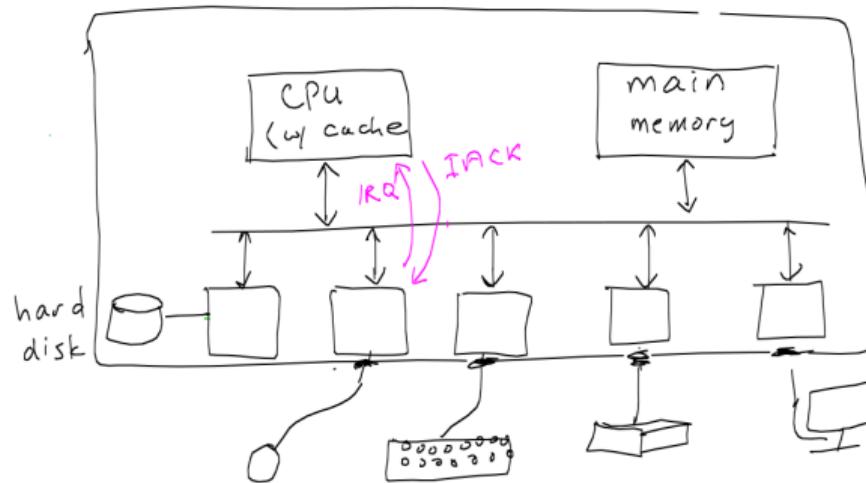


Esquemas de I/O: Interrupções

- 1) each I/O device has its own IRQ, IACK lines
(similar in flavour to BR/BG in DMA)

Disadvantages:

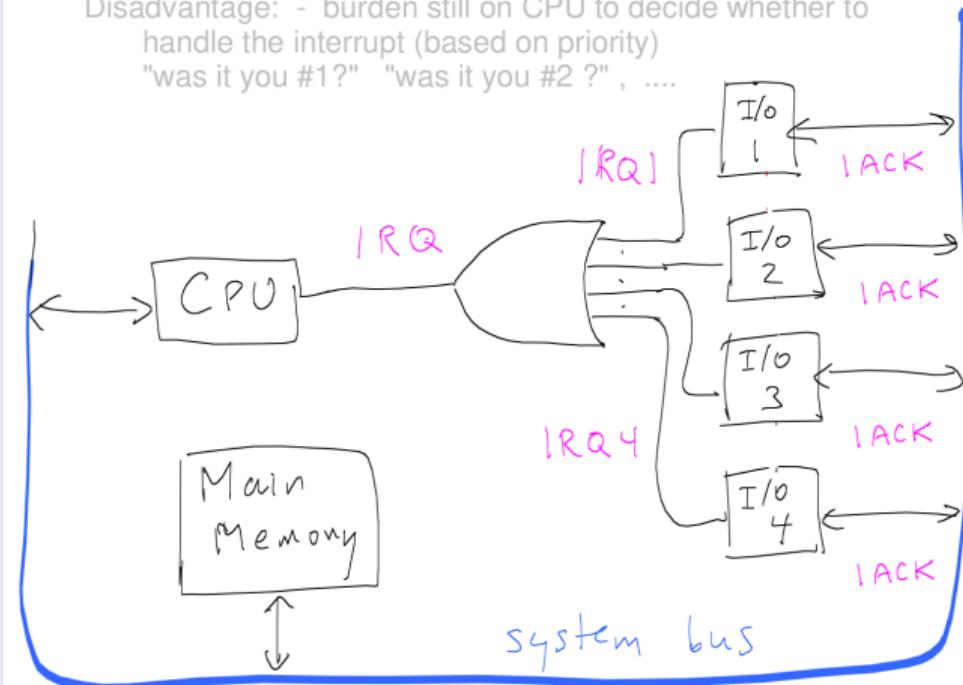
- many lines
- places burden entirely on CPU to decide on priorities



Esquemas de I/O: Interrupções

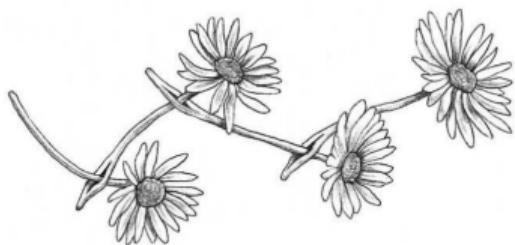
2.) shared IRQ + polling

Disadvantage: - burden still on CPU to decide whether to handle the interrupt (based on priority)
"was it you #1?" "was it you #2 ?" , ...



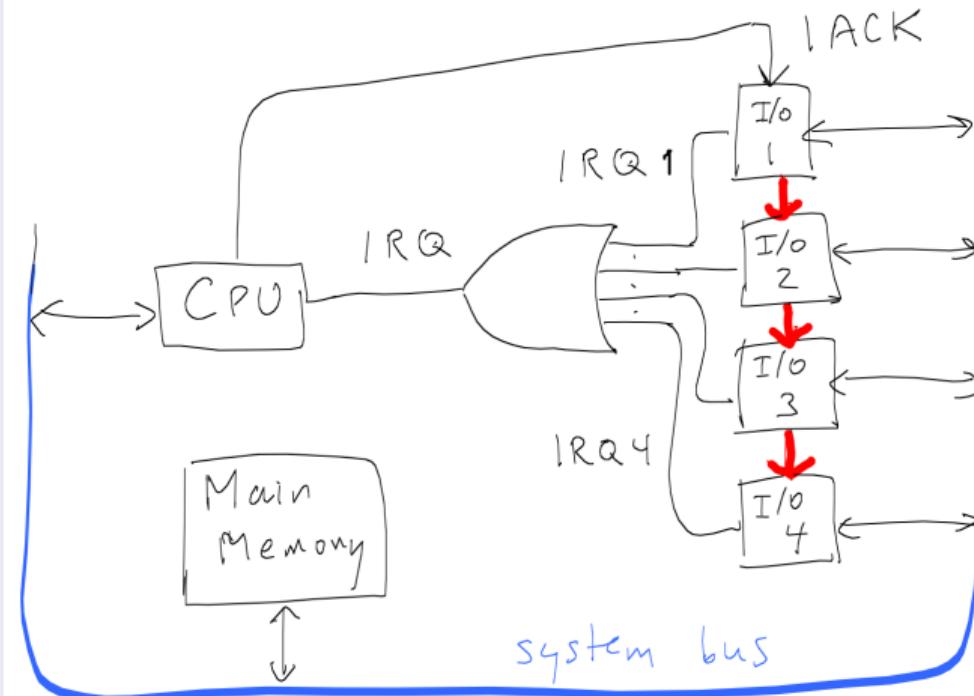
Esquemas de I/O: Interrupções

3. Daisy Chain



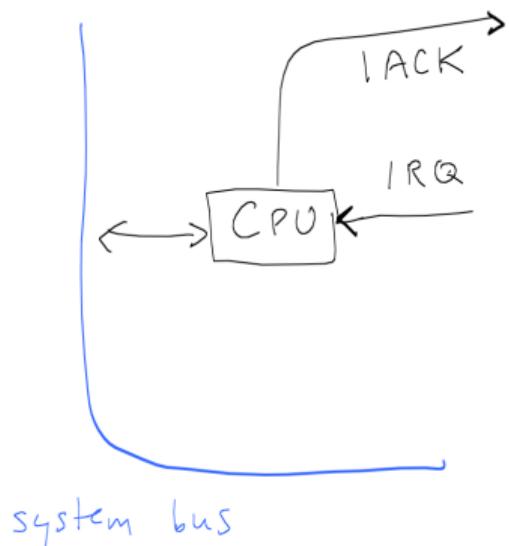
Esquemas de I/O: Interrupções

3.) Daisy Chain (priority ordering = physical ordering)



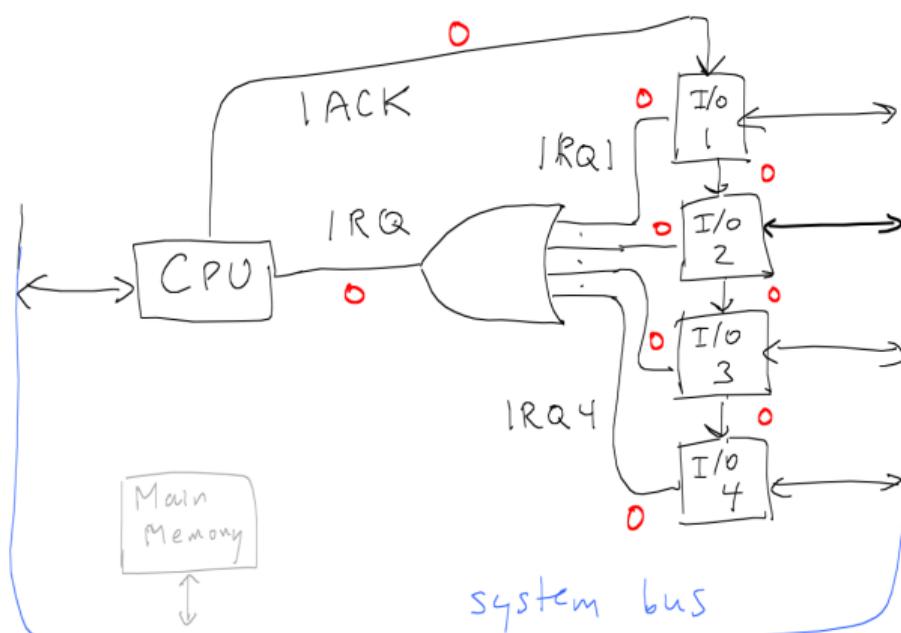
Esquemas de I/O: Interrupções

The key advantage of daisy chaining is that the I/O controllers (not the CPU) decide on priorities of interrupt requests. HOW ?



If the CPU is available then it allows the I/O device to use the bus to make an interrupt request.

Suppose all IRQ signals are initially 0.

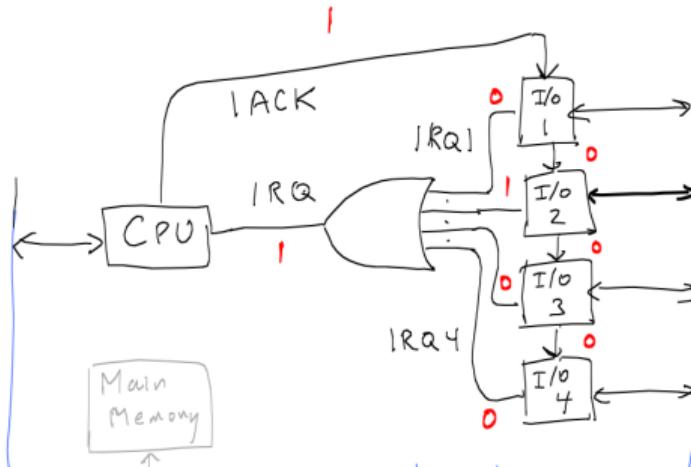


Esquemas de I/O: Interrupções

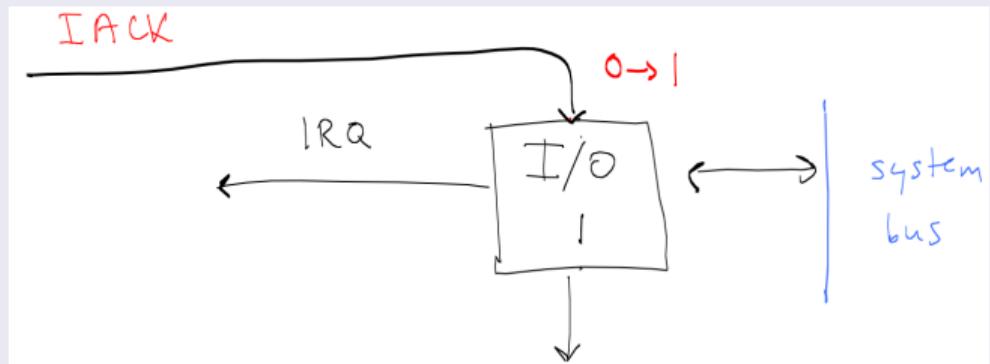
Suppose IRQ 2 makes an interrupt request

IRQ 2 = 1 : "knock, knock"

IACK = 1 : "who is there?"
(and CPU frees up system bus temporarily)



Esquemas de I/O: Interrupções



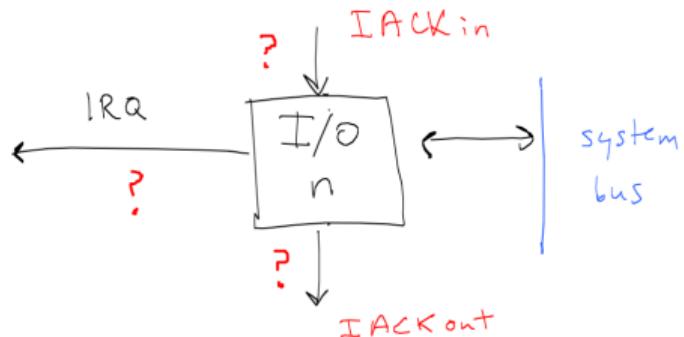
I/O device 1 observes **IACK** change from 0 to 1.

If device 1 made the interrupt request,
then it uses the system bus to communicate with CPU
(and it passes a 0 to the next device controller).

else it sets to 1 the signal to the next I/O device controller.

Esquemas de I/O: Interrupções

More generally... I/O device controller n



I/O device n observes its **IACK in** change from 0 to 1.

If device n made the interrupt request, then it sets its **IACK out** to 0, and uses the system bus to communicate with CPU.

Otherwise it sets its **IACK out** to 1.

Esquemas de I/O: Interrupções

IRQ = 1 : "knock, knock" (recall OR gate)

IACK = 1 (CPU) : "who is there?"
(and CPU frees up system bus)

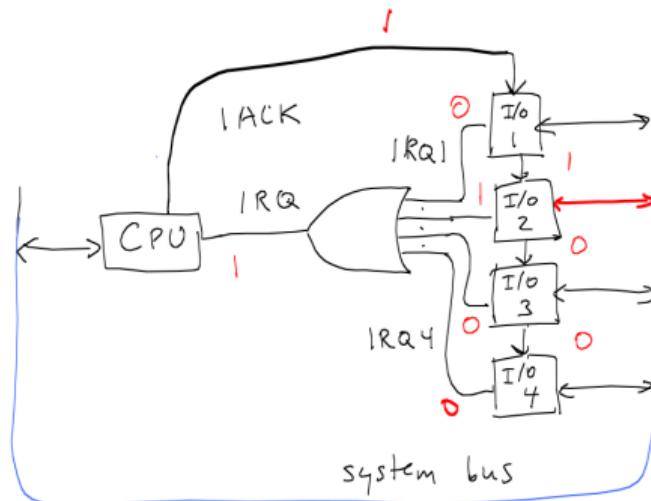
IACK gets passed down to highest priority device that initiated IRQ.

I/O device n : "its me, device n" (on system bus)

IACK (CPU) = $\begin{cases} 0, & \text{"go away, you are not important enough"} \\ 1, & \text{"yes, what can I do for you?"} \end{cases}$

Esquemas de I/O: Interrupções

A higher priority device **can** interrupt a lower priority device.
How ?



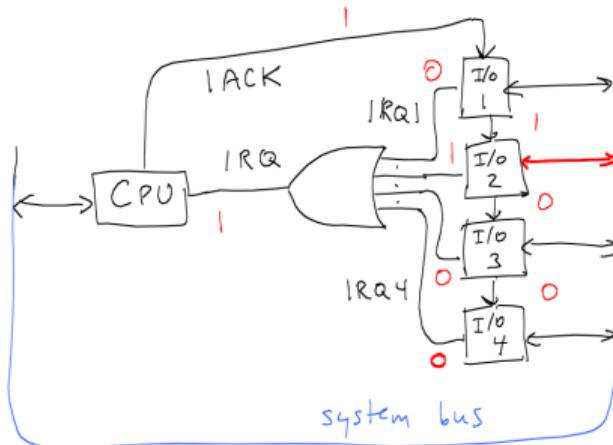
Q: How does I/O device 1 interrupt I/O device 2 ?

A: I/O device 1 sets its IACK out to 0

I/O device 2 wraps up and sets IRQ2 to 0. Tries again

Esquemas de I/O: Interrupções

A lower priority device **cannot** interrupt a lower priority device.
Why not ?

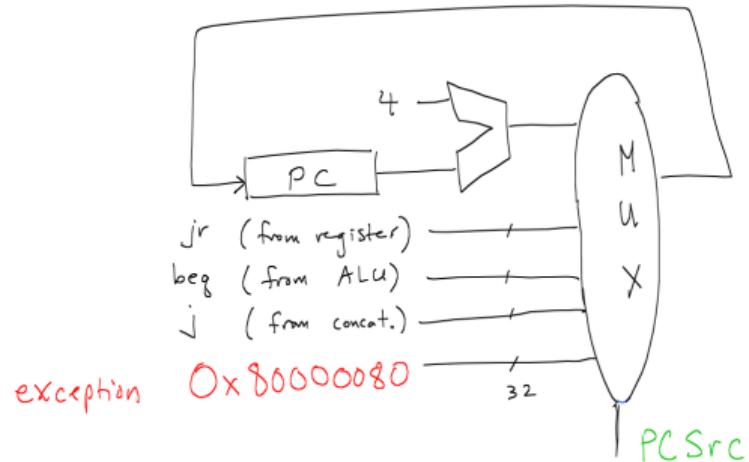


Q: Why can't I/O device 3 interrupt I/O device 2 ?

A: IRQ3 goes through the same OR gate as IRQ2.

I/O is not allowed to write onto system bus until it sees
IACK in go from 0 to 1 (indicating bus has been freed up)

Esquemas de I/O: Interrupções



Program jumps to **exception handler** in the kernel.

Here we have a more general notion of interrupt, which can include both external interrupts (I/O, just discussed) and internal interrupts ("exceptions" such as overflow, division by 0, bad Address, timer = 0).

Esquemas de I/O: Interrupções

Interrupt handler

Decide if the interrupt should be handled (sketched in a few slides)

If yes,

- disable *all* interrupts
- save state of current process
(store register values in kernel memory)
- enable higher priority interrupts only
- service the interrupt
- possibly run other processes
- restore the state of the process and return to it

Esquemas de I/O: Interrupções

Note there are TWO reasons why an interrupt might be *ignored*:

- the CPU is currently handling a higher priority interrupt
- the CPU is handling a lower priority interrupt BUT it needs to finish saving state information, before that interrupt itself can be interrupted.



Esquemas de I/O: Interrupções

| Registers | Coproc 1 | Coproc 0 |
|---------------|----------|------------|
| Name | Number | Value |
| \$8 (vaddr) | 8 | 0x00000000 |
| \$12 (status) | 12 | 0x0000ff11 |
| \$13 (cause) | 13 | 0x00000000 |
| \$14 (epc) | 14 | 0x00000000 |

In particular, \$12 is the status register



bit 0 of \$12 is
"interrupt enable"



MIPS registers

| Name | Register Number | Usage | Preserved on call |
|-----------|-----------------|-----------------------------------|-------------------|
| \$zero | 0 | the constant value 0 | n.a. |
| \$at | 1 | reserved for the assembler | n.a. |
| \$v0-\$v1 | 2-3 | value for results and expressions | no |
| \$a0-\$a3 | 4-7 | arguments (procedures/functions) | yes |
| \$t0-\$t7 | 8-15 | temporaries | no |
| \$s0-\$s7 | 16-23 | saved | yes |
| \$t8-\$t9 | 24-25 | more temporaries | no |
| \$k0-\$k1 | 26-27 | reserved for the operating system | n.a. |
| \$gp | 28 | global pointer | yes |
| \$sp | 29 | stack pointer | yes |



Esquemas de I/O: Interrupções

```
# enable interrupts

mfco $k0, $12 # $12 is Status (on coprocessor 0)
ori $k0, $k0, 0x01 # turn on only LSB
mtc0 $12, $k0

# disable interrupt

mfco $k0, $12
lui $1, 0xffff
ori $1, $1, 0xffffe (turn off LSB of $12)
and $k0, $k0, $1
mtc0 $12, $k0
```



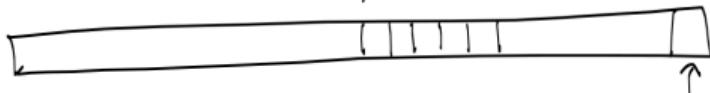
Esquemas de I/O: Interrupções

| Registers | Coproc 1 | Coproc 0 |
|---------------|----------|------------|
| Name | Number | Value |
| \$8 (vaddr) | 8 | 0x00000000 |
| \$12 (status) | 12 | 0x0000ff11 |
| \$13 (cause) | 13 | 0x00000000 |
| \$14 (epc) | 14 | 0x00000000 |

status register enables/disables different priority interrupts



cause register specifies which priority interrupt has occurred



Interfaceamento

- Fabricantes tem liberdade de definir a arquitetura das interconexões internamente a um sistema eletrônico (CPU)
- Para interligar esses sistemas eletrônicos na solução de um problema da vida real, um conjunto de regras de construção e sinalização dos barramentos deve existir (afinal, vários participantes querem usar o barramento para trocar mensagens): protocolos de barramentos.
- São considerados aspectos mecânicos (físicos), elétricos e lógicos
- Alguns barramentos:
 - IBM PC bus (PC/XT)
 - ISA bus (PC/AT)
 - Microchannel (PS/2)
 - PCI bus
 - SCSI bus
 - USB
 - FireWire
 - PCI Express

Exemplos: PCI (Peripheral Component Interconnect Bus)

Barramento primário para IO em computadores

Antecessores:

- ISA (8.33 MHz, 2B/ciclo, 16.7 MB/s)
- EISA (8.33 MHz, 4B/ciclo, 33.3 MB/s)

Demandas por barramentos de alta velocidade quando o PC passou a ser plataforma para aplicações multimedia:
155 MB/s



Exemplos: PCI (Peripheral Component Interconnect Bus)

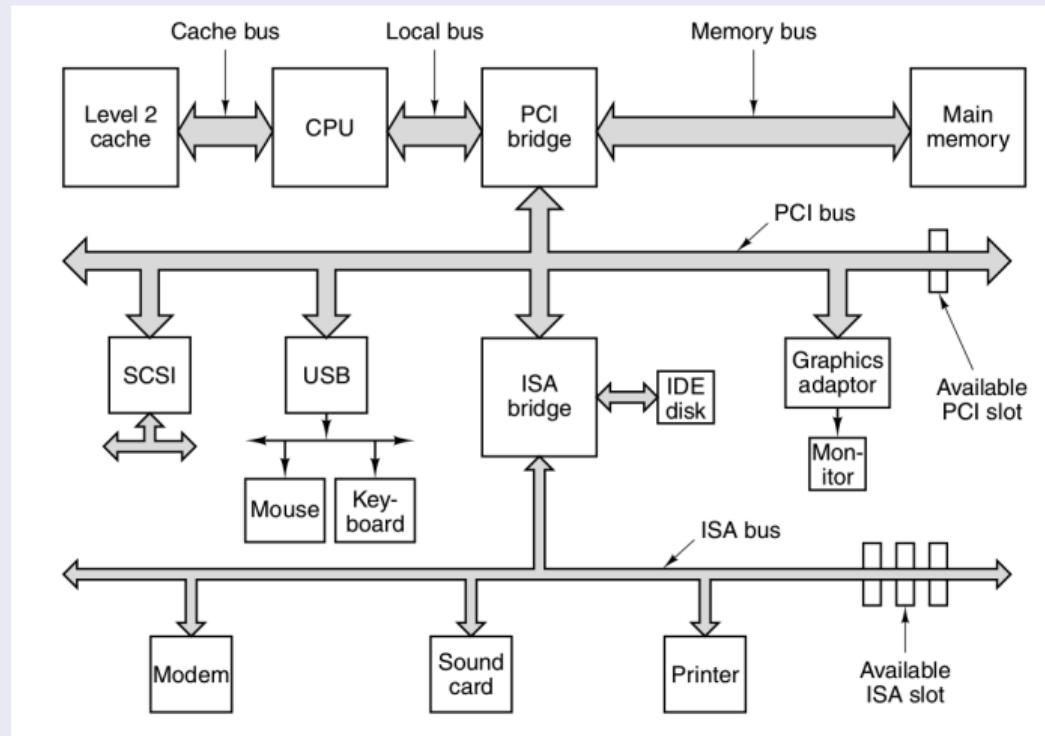
Duas versões: tradicional (1990, Intel) e o PCIe

- padrão aberto: Intel abriu mão das patentes
- 33 MHz, 32 bits (1 word) por ciclo \Rightarrow 133 MB/s
- 66 MHz, 64 bits (1 word) por ciclo \Rightarrow 528 MB/s
- Barramento síncrono



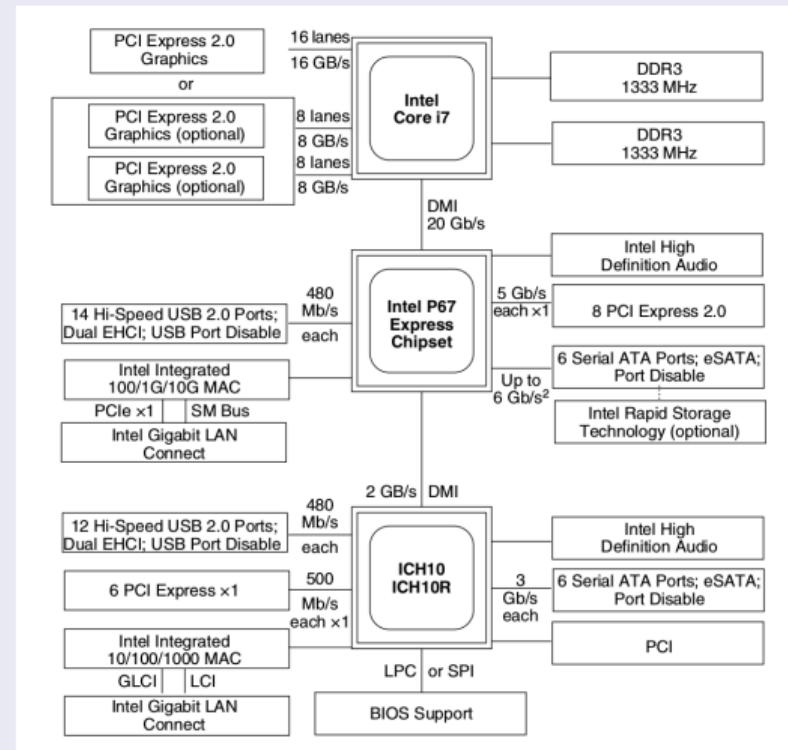
Exemplos: PCI (Peripheral Component Interconnect Bus)

Arquitetura clássica:



Exemplos: PCI (Peripheral Component Interconnect Bus)

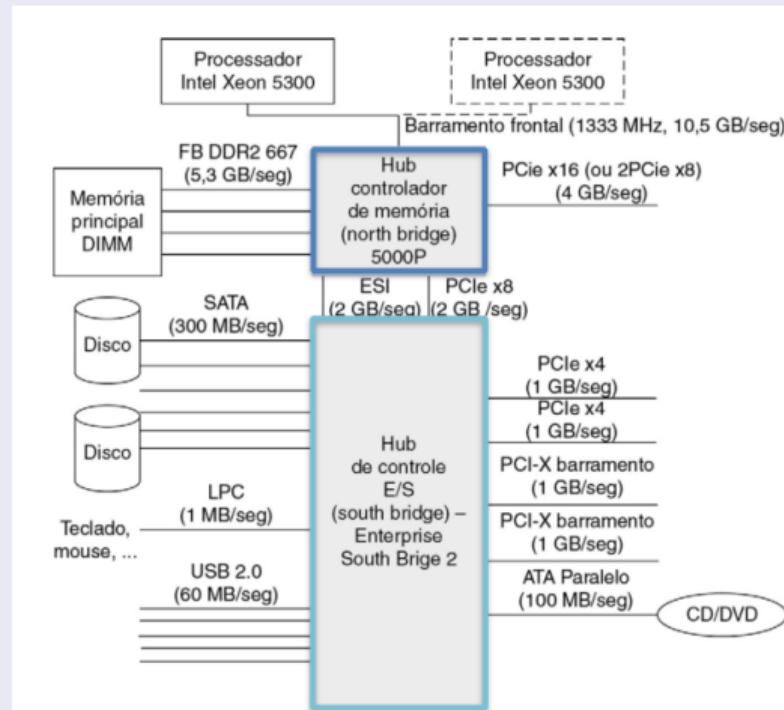
Arquitetura contemporânea:



Barramentos

Exemplos: PCI (Peripheral Component Interconnect Bus)

Arquitetura contemporânea:



Exemplos: PCI (Peripheral Component Interconnect Bus)

Problemas:

- Não retrocompatível com antecessores.
- Apesar da velocidade, não era capaz de sustentar o desempenho esperado de um barramento para interligação da CPU com a memória principal.
- Demandou arquitetura multibarramento: um barramento de altíssima velocidade para interligação da CPU com elementos de tempo crítico e outros barramentos de interconexão com periféricos.



Exemplos: PCIe (PCI Express)

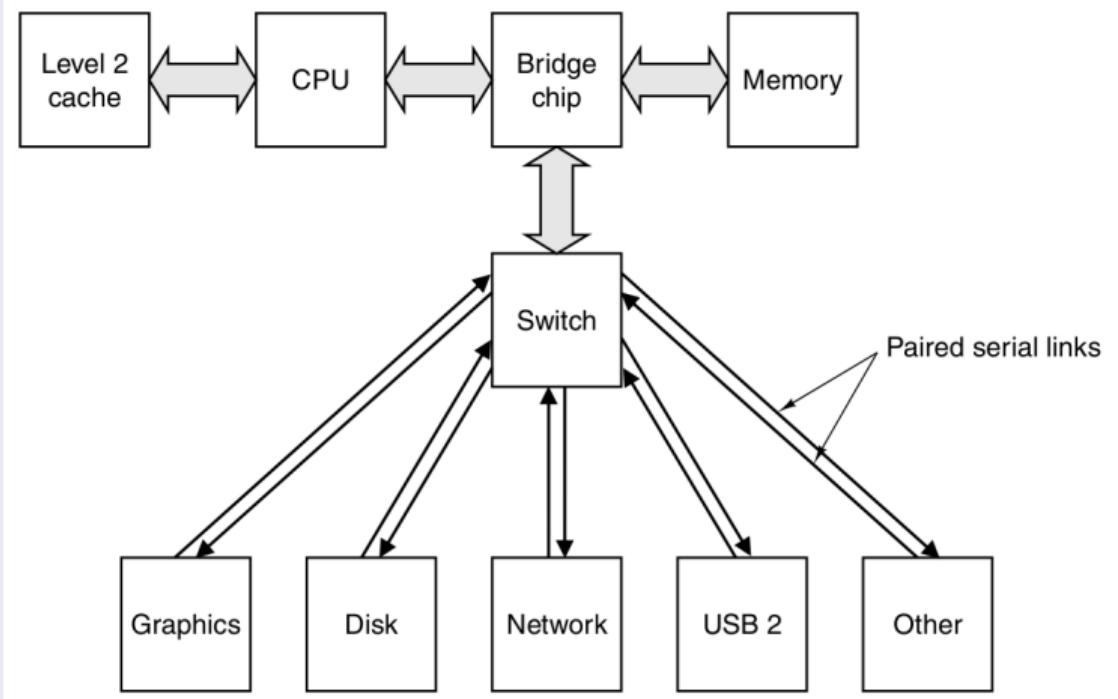
Atender demanda por mais largura de banda para sustentar IO:

Arquitetura

- substituir barramentos paralelos com muitos mestres e escravos (e seus problemas de controle/arbitragem)
- projeto baseado em **linhas de comunicação seriais ponto-a-ponto**: comutador que interliga os periféricos à CPU
- comunicação baseada em comutação de pacotes!
- códigos corretores de erro para proteger os dados trocados através das linhas
- conexão fisicamente curta: compatibilidade/imunidade eletromagnética
- expansível: cascata de switches (comutadores)
- **hot-plug!**
- conectores pequenos



Exemplos: PCIe (PCI Express)



Exemplos: USB (Universal Serial Bus)

Barramentos ora estudados:

- PCI e PCIe: excelentes para dispositivos de elevada velocidade, porém muito caro para dispositivos de IO de baixa velocidade como teclados e mouses.

Realidade: cada periférico a ser conectado ao PC demandava a abertura do gabinete e configurações (físicas e lógicas): jumpers, dip-switches, bios, drivers, conflitos de IRQ, reboot, ...



Exemplos: USB (Universal Serial Bus)

Objetivos perseguidos com a proposta do USB

- usuários não deverão ajustar dip-switches ou jumpers nas placas ou dispositivos
- usuários não deverão abrir o gabinete para a instalação de novos dispositivos de IO
- deverá haver apenas um tipo de cabo para conectar todos os dispositivos
- os dispositivos de IO deverão receber sinal de força pelo cabo
- até 127 dispositivos poderão ser conectados a um computador
- o sistema deverá suportar dispositivos de tempo-real (telefone, altofalantes, ...)
- os dispositivos poderão ser instalados enquanto o computador estiver em execução
- nenhuma reinicialização será necessária depois das instalação de um novo dispositivo
- o novo barramento e seus dispositivos de IO deverão ser viáveis (em termos de custo) para manufatura.



Exemplos: USB (Universal Serial Bus)

Versões:

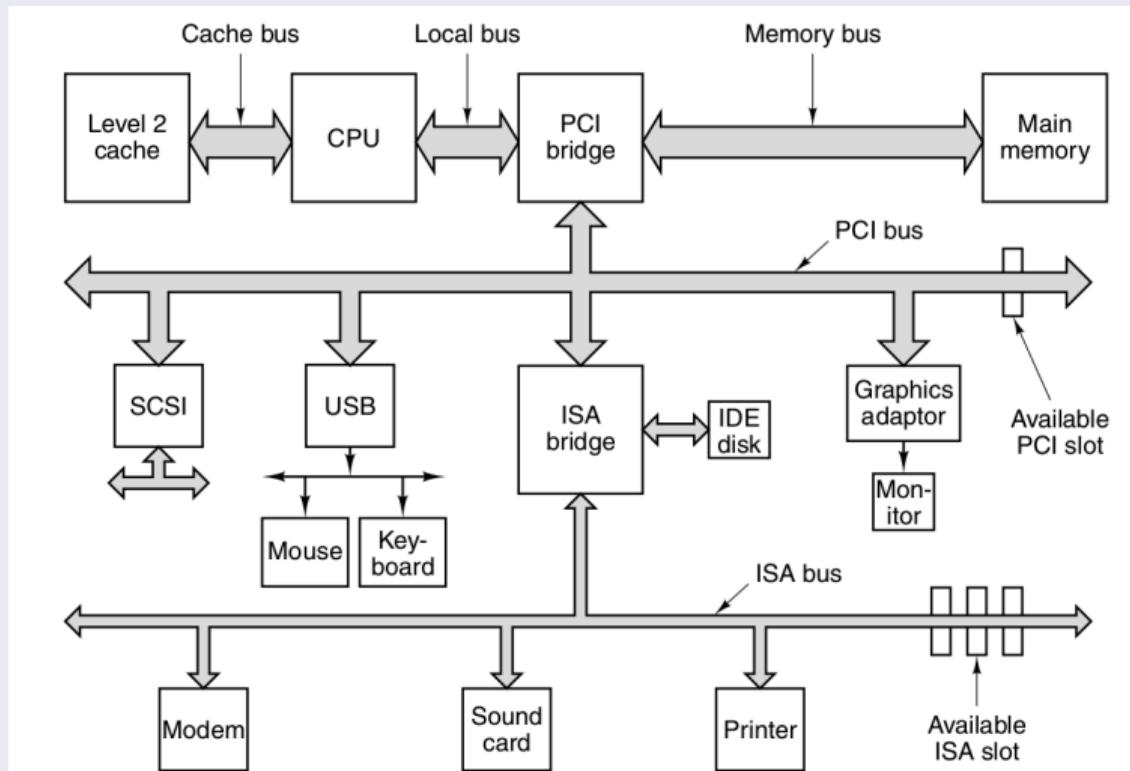
- 1.0: 1.5 Mbps
- 1.1: 12 Mbps
- 2.0: 480 Mbps
- 3.0: 5 Gbps

Arquitetura:

- root hub: conectado ao barramento principal do computador
- o root hub possui conectores para cabos que serão usados na conexão aos dispositivos de IO ou a hubs de expansão.
- conectores dos cabos possuem formatos físicos bem distintos para evitar conexões acidentais



Exemplos: USB (Universal Serial Bus)



Exemplos: USB (Universal Serial Bus)

Cabo:

- 4 linhas
- 2 de dados
- 2 de força/potência: +5 V e GND

