Introdução ao MIPS

– *Input/Output*, *Polling* e Interrupções –

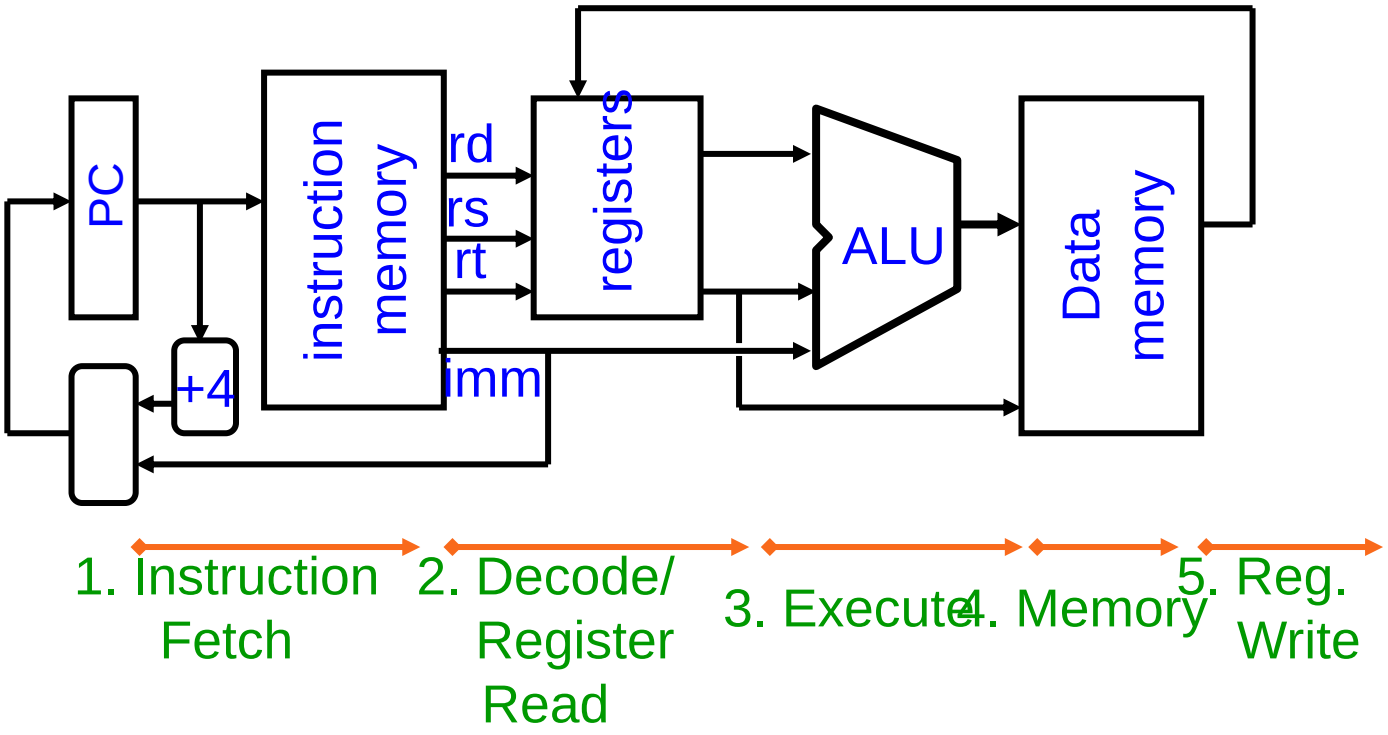
Arquitetura de Computadores 2024/2025

Objectivos da Aula de Hoje

- Como o processador interage com o seu ambiente?
  - Panorâmica sobre a unidade de entrada/saída (I/O)
- Como trocar informação com os dispositivos?
  - I/O Programada ou I/O Mapeado em Memória
- Como lidar com eventos?
  - *Polling* ou Interrupções
- Como transferir grandes quantidades de dados?
  - Acesso Directo à Memória (*DMA Direct Memory Access*)

Como o processador interage com o meio ambiente?

Datapath do MIPS



Computer System Organization = Memory +Datapath +Control + Input +Output

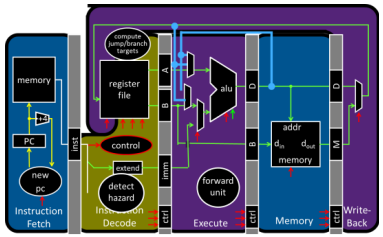
Os dispositivos de I/O possibilitam a interacção com o meio-ambiente:

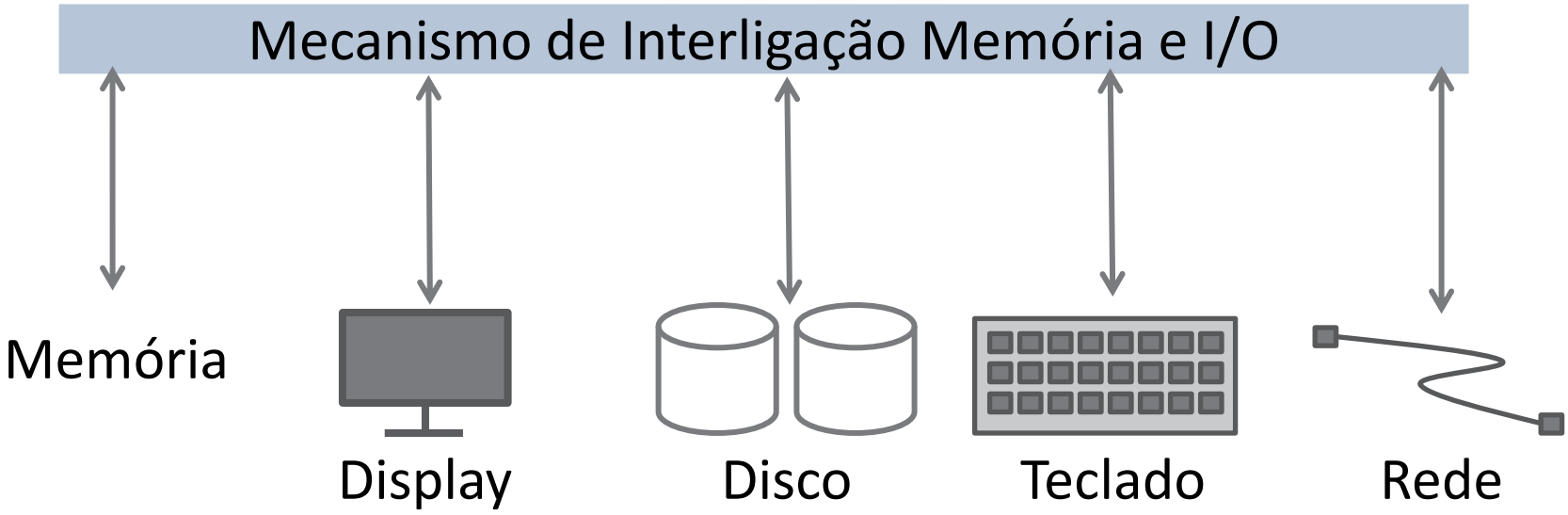
Dispositivo	Tipo	Agente	*Data Rate* (b/sec)
Keyboard	Input	Human	100
Mouse	Input	Human	3.8k
Sound Input	Input	Machine	3M
Voice Output	Output	Human	264k
Sound Output	Output	Human	8M
Laser Printer	Output	Human	3.2M
Graphics Display	Output	Human	800M – 8G
Network/LAN	Input/Output	Machine	100M – 10G
Network/Wireless LAN	Input/Output	Machine	11 – 54M
Optical Disk	Storage	Machine	5 – 120M
Flash memory	Storage	Machine	32 – 200M
Magnetic Disk	Storage	Machine	800M – 3G

Tentativa #1: Mecanismo Único de Interligação

Implicações: substituir todos os dispositivos sempre que o mecanismo de interligação mude.

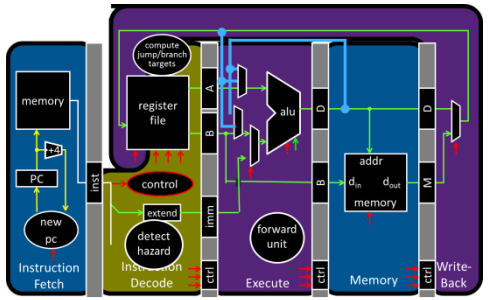
Velocidade do teclado == Velocidade da Memória Principal?

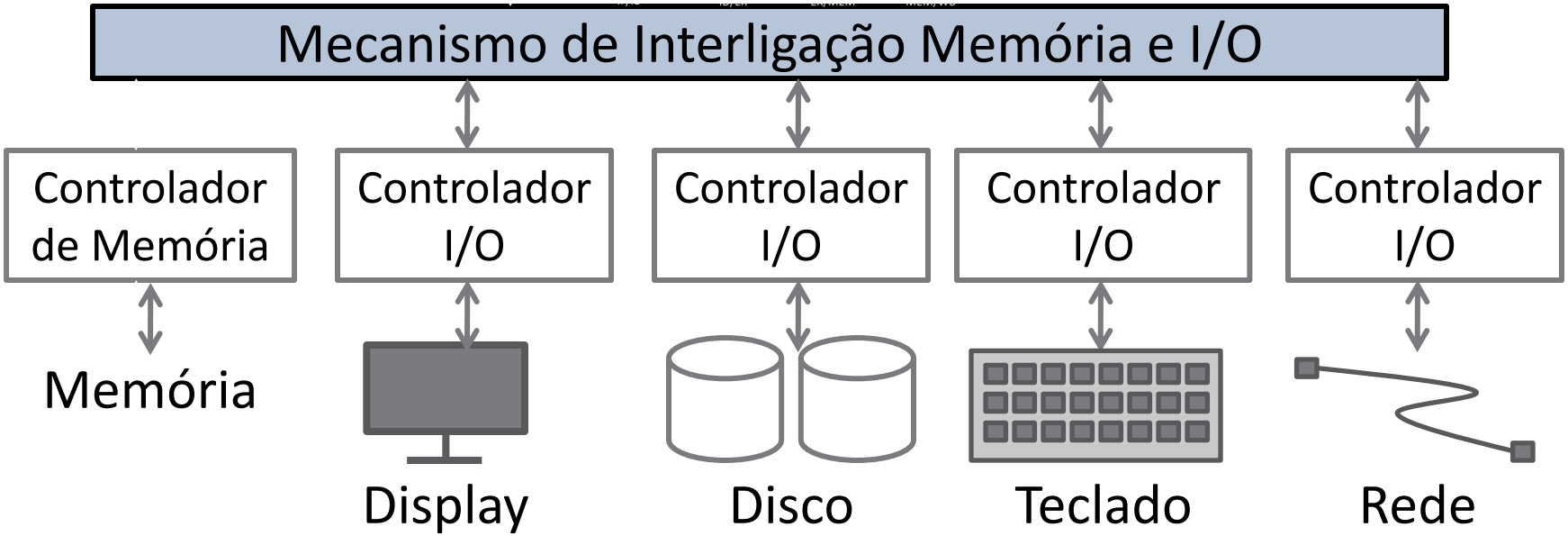




Tentativa #2: Controladores de I/O

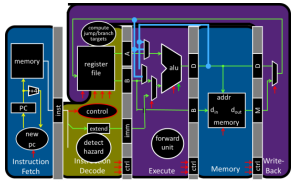
- Desacoplar Dispositivos de I/O do mecanismo de interligação
- Permitir desenhar interfaces I/O interfaces mais inteligentes

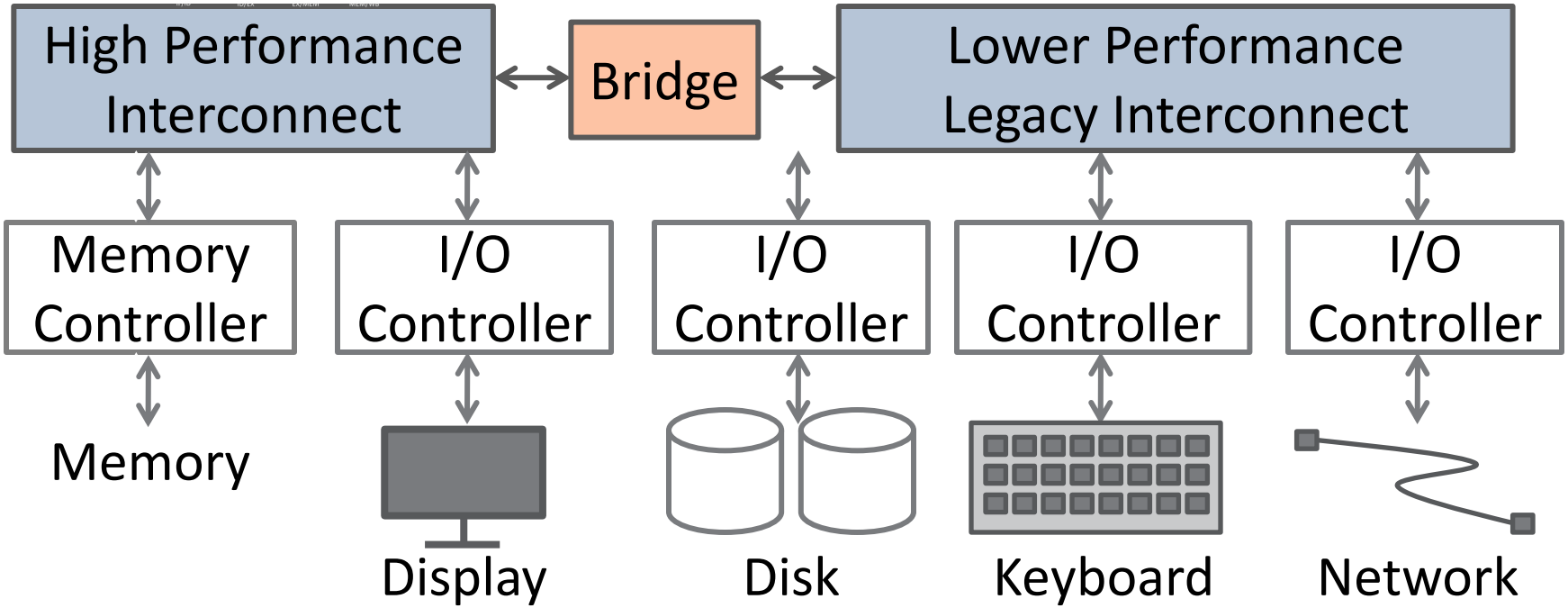




Tentativa #3: Controladores I/O + *Bridge*

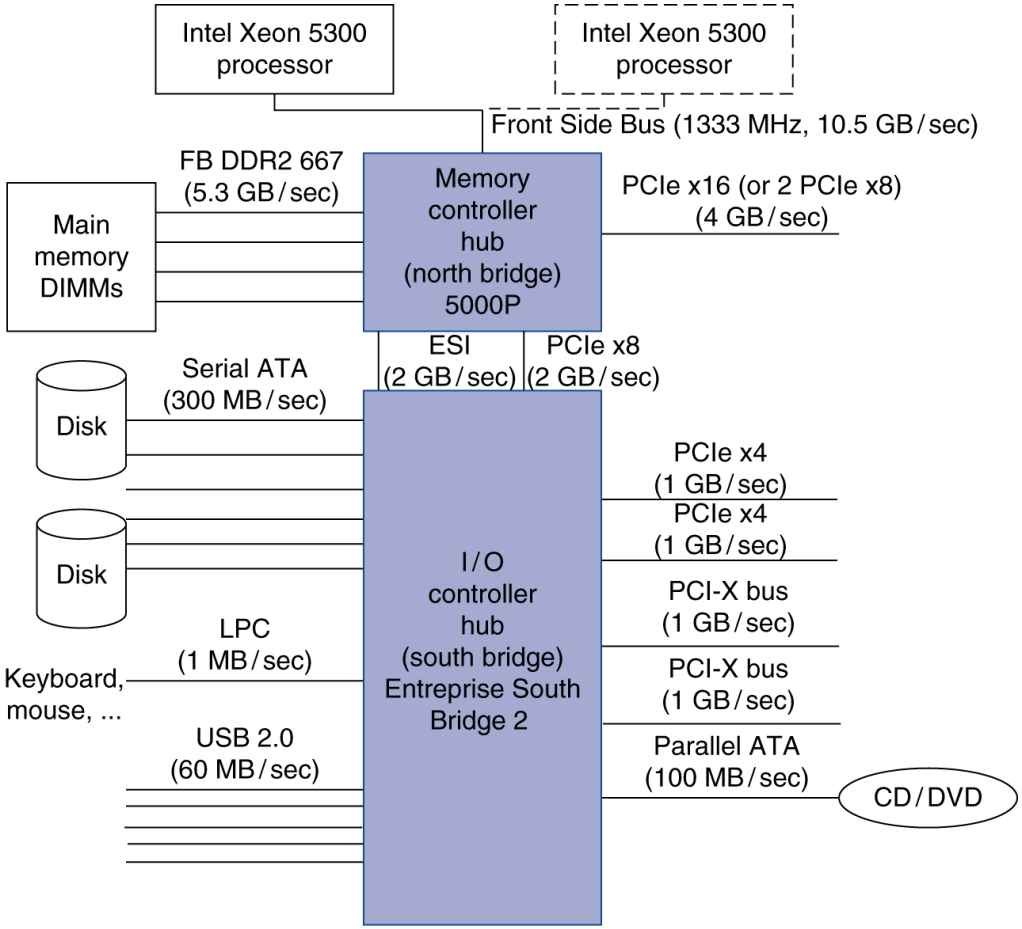
- Separar os componentes de alta velocidade como o processador, a memória e a placa gráfica dos dispositivos de baixa velocidade:





Tentativa#3: Controladores I/O + *Bridge*

Separar os componentes de alta velocidade como o processador, a memória e a placa gráfica dos dispositivos de baixa velocidade:



Parâmetros de *Bus*

- Largura = Número de Fios
- Tamanho da Transferência = Nº de bytes por transação
- Taxa da Transferência = Nº máximo de bytes transacionados por unidade de tempo
- Síncrono (utilizando o relógio do *bus*) ou Assíncrono (sem utilizar o relógio do bus / "relógio próprio")

Tipos de Barramentos (*Bus*)

- Bus Processador – Memória (“*Front Side Bus, QPI*)
  - Curto, rápido e largo
  - Topologia tipicamente fixa, desenhada por “*chipset*”
    - CPU + Caches + Interligações + Controlador de Memória
- Bus Periféricos e I/O (PCI, SCSI, USB, LPC, …)
  - Mais comprido, mais lento e mais estreito
  - Topologia flexível, ligações múltiplas/variadas
  - Interoperabilidade entre vários dispositivos
  - Liga-se ao bus processador-memória através de uma ponte (*bridge*)

Exemplos de *Buses* (Interligações)

Nome	Tipo	Nº Dispositivos por Canal	Largura do Canal	*Data Rate* (B/sec)
Firewire 800	External	63	4	100M
USB 2.0	External	127	2	60M
Parallel ATA	Internal	1	16	133M
Serial ATA (SATA)	Internal	1	4	300M
PCI 66MHz	Internal	1	32-64	533M
PCI Express v2.x	Internal	1	2-64	16G/dir
Hypertransport v2.x	Internal	1	2-64	25G/dir
QuickPath (QPI)	Internal	1	40	12G/dir

Interligação de Componentes

- Módulos de interligação são (eram?) *buses*
  - Conjunto de fios paralelos para dados e sinais de controle
  - Canais partilhados
    - múltiplos emissores/receptores
    - Todos os componentes podem ver as transações no *bus*
  - Protocolo de bus: conjunto de regras para gerir a utilização do *bus*

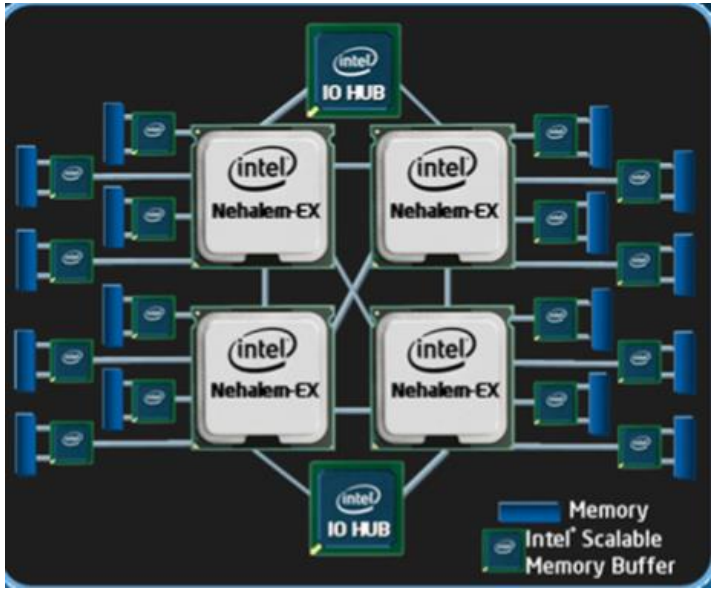
Ex. Intel Xeon

- Alternativas (cada vez mais comuns):
  - Canais dedicados ponto-a-ponto

Ex. Intel Nehalem

Tentativa #4: Controladores I/O + Bridge + NUMA

- Remove as *bridge* uma vez que estas apresentam problemas de congestionamento de tráfico com as interligações ponto-a-ponto
- Ex. Acesso à Memória não Uniforme (*Non-Uniform Memory Access, NUMA*)



Resumo

Unidades de I/O diferentes requerem uma organização hierárquica das interligações entre componentes. A tendência é cada vez mais optar-se por uma topologia com ligações ponto-a-ponto para privilegiar a velocidade.

A seguir…

Como é que o processador interage com os dispositivos de entrada/saída?

Interfaces I/O Baseados em *Device Drivers*

Conjunto de métodos que permitem escrever/ler dados de/para os dispositivos e respectivas unidades de controlo

Exemplo: ***Linux Character Devices***

```
// Open a toy " echo " character device
int fd = open("/dev/echo", O_RDWR);

// Write to the device
char write_buf[] = "Hello World!";
write(fd, write_buf, sizeof(write_buf));

// Read from the device
char read_buf [32];
read(fd, read_buf, sizeof(read_buf));

// Close the device
close(fd);

// Verify the result
assert(strcmp(write_buf, read_buf)==0);
```

APIs para dispositivos de I/O

Estrutura Típica de uma API de um dispositivo de I/O, um conjunto de registos *read-only* ou *read/write*

Registos de Comando

– Uma escrita neste registo permite o dispositivo realizar uma dada tarefa

Registos de Estado

– Leitura permite indicar o que está a ser feito, códigos de erro, etc.

Registos de Dados

– Escrita: transfere dados para o dispositivo
– Leitura: transfere dados do dispositivo

I/O Device API

- Exemplo simples (antigo): Teclado

8-bit Status (R): PE TO AUXB LOCK AL2 SYSF IBS OBS

8-bit Command (W):

0xAA = "*self test*"

0xAE = "*enable kbd*"

0xED = "*set LEDs*"

…

8-bit Data (R/W):

scancode (quando lido)

LED state (quando escrito)…

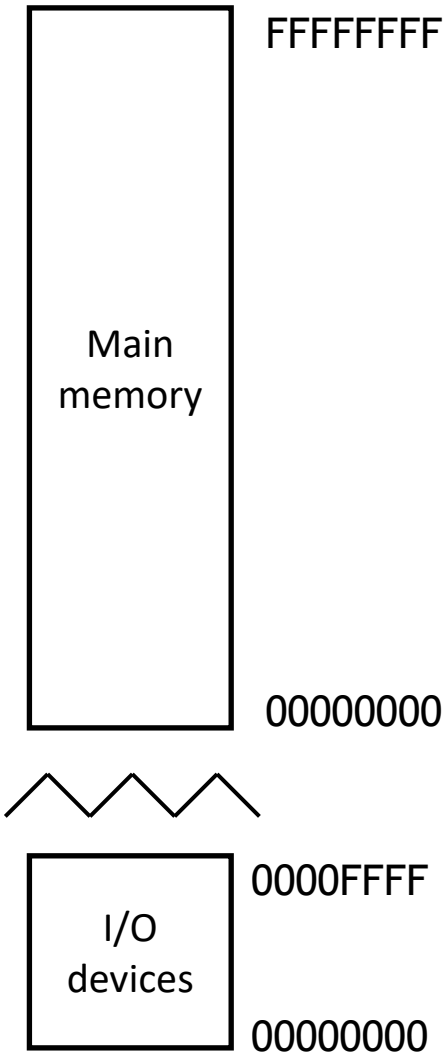
Interfaces de Comunicação 1/2

- Q: Como pode o processador comunicar com o dispositivo?
- R: Introdução de instruções especiais para I/O
- I/O Programado ← Interage directamente com os registos de estado, dados e comando
  - inb $a, 0x64 ← registo de estado do teclado
  - outb $a, 0x60 ← registo de dados do teclado
  - Especifica: dispositivo, dados, direcção de transferência
  - Protecção: instruções apenas permitidas em modo de kernel

*x86: $a implícito; também existe inw, outw, inh, outh, …

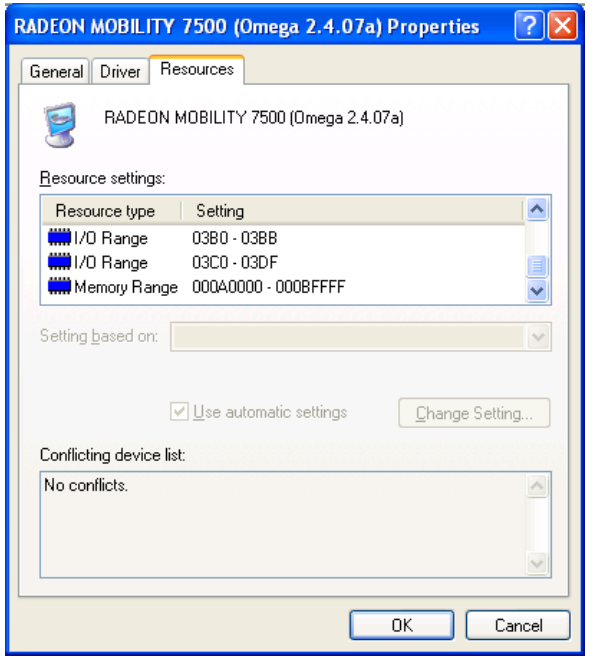
I/O Programado

- Assume a existência de um espaço de endereçamento separado para acesso à memória e aos dispositivos externos.
- Por exemplo, a família de microprocessadores da Intel têm um espaço de endereçamento de 32-bits.
  - As instruções normais como o MOV referenciam a memória principal.
  - Instruções especiais como o IN e o OUT permitem o acesso a um espaço de endereçamento separado para I/O de 64KB.



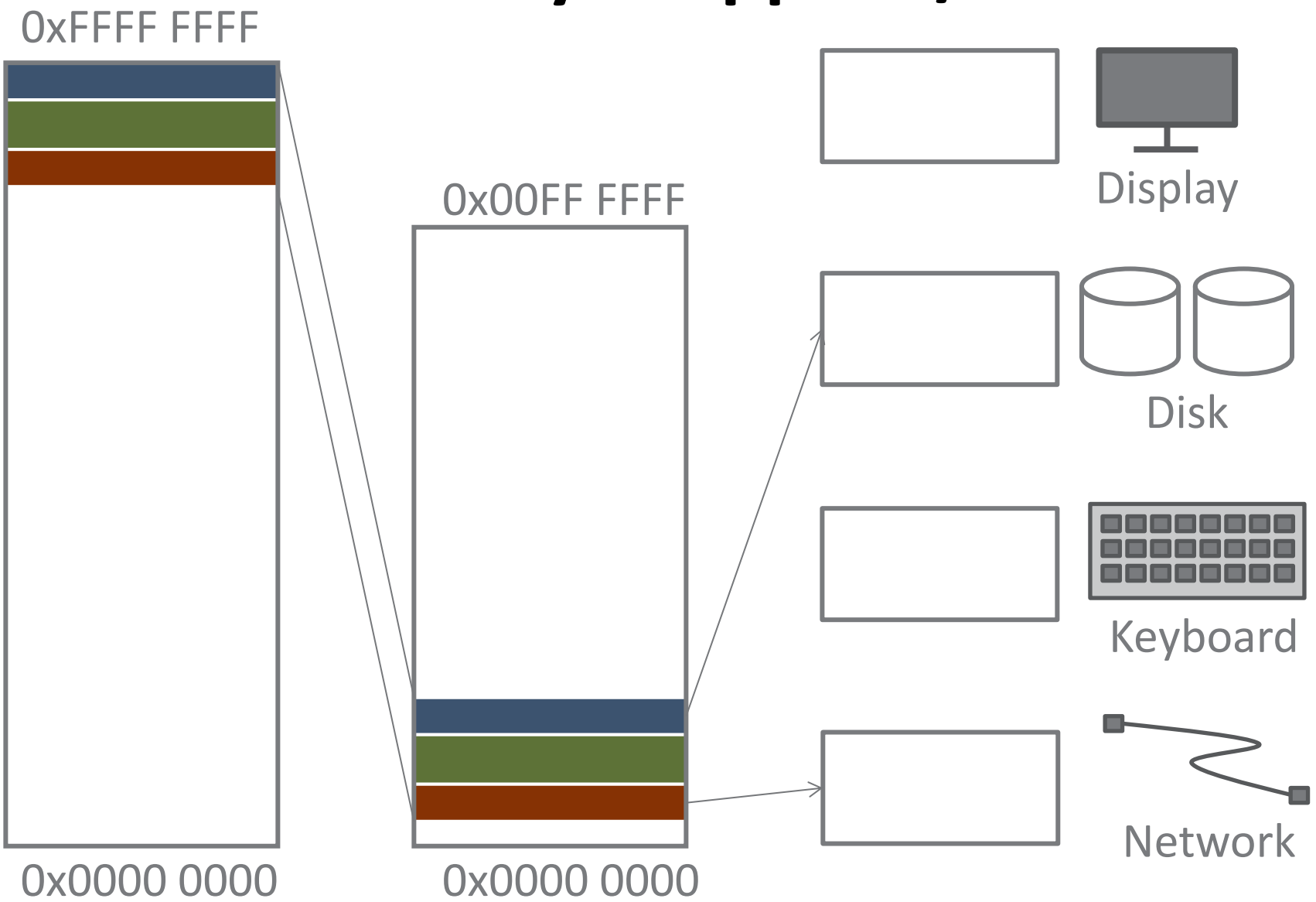
Interfaces de Comunicação 2/2

- Q: Como pode o processador comunicar com o dispositivo?
- A: Mapear registos num espaço de endereçamento virtual



- Memory-mapped I/O ← Mais rápido
  - O acesso a certos endereços de memória são redirecionados para os dispositivos
  - Os dados circulam no bus da memória

Memory-Mapped I/O



Device Drivers

Programmed I/O

```
char read_kbd()
{
do {
    sleep();
    status = inb(0x64);
  } while(!(status & 1));

  return inb(0x60);
}
```

syscall

Memory Mapped I/O

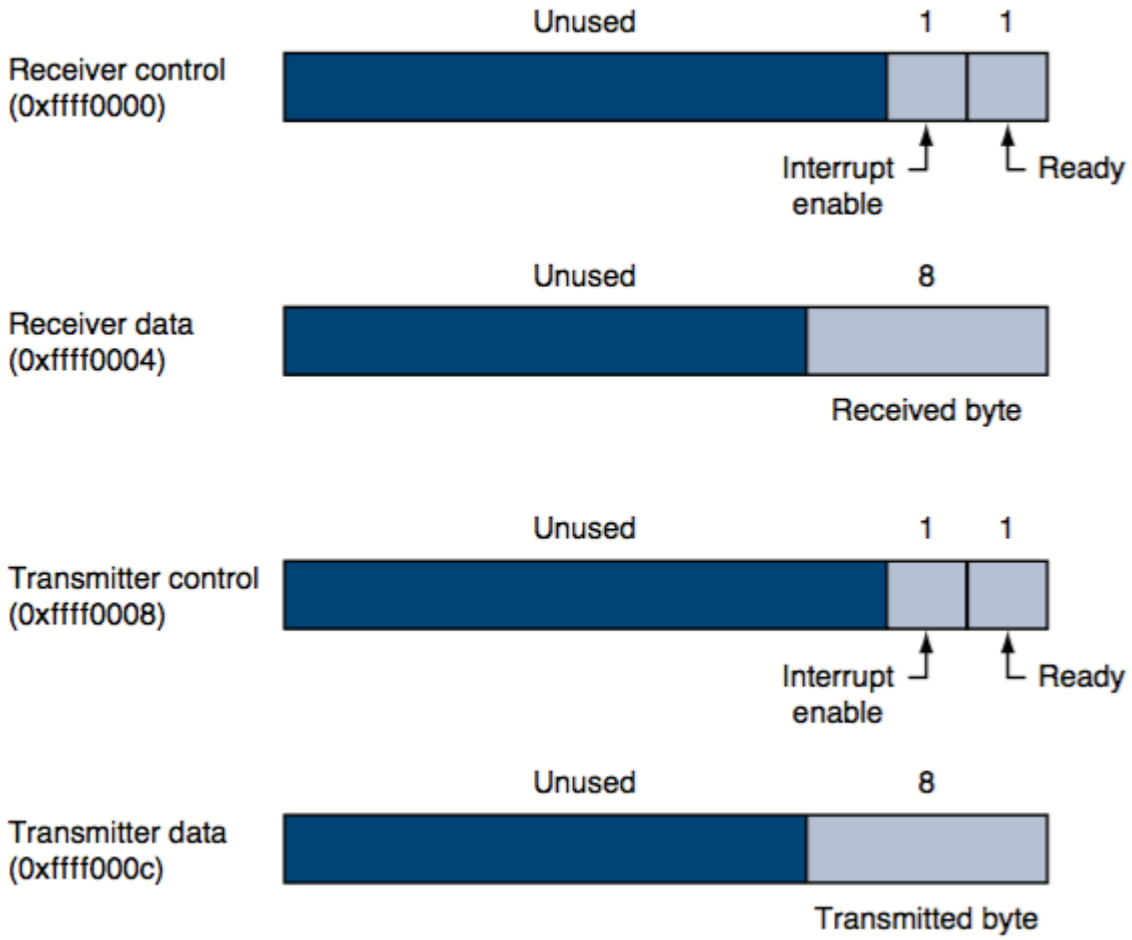
```
struct kbd {
  char status, pad[3];
  char data, pad[3];
};
kbd *k = mmap(...);

char read_kbd()
{
  do {
    sleep();
    status = k->status;
  } while(!(status & 1));
  return k->data;
}
```

syscall

NO syscall

Exemplo: *Memory-mapped terminal device*



Comparação entre I/O Programado vs Memory Mapped I/O

- I/O Programado
  - Requer instruções especiais
  - Pode requerer hardware dedicado para fazer a interface com os dispositivos
  - Mecanismos de proteção via acesso restrito ao *kernel* às instruções de I/O
  - A virtualização pode ser difícil

- *Memory-Mapped* I/O
  - Utiliza as instruções normais de *load/store*
  - Utiliza as interfaces standard com a memória
  - Mecanismos de protecção à custa dos esquemas de proteção de memória
  - A Virtualização é possível através dos esquemas normais de virtualização de memória

Métodos de Comunicação I

- Como é que o programa sabe se o dispositivo está pronto ou terminou a tarefa?
- Polling: Verificar periodicamente o registo de estado
  - *If device ready, do operation*
  - *If device done, …*
  - *If error, take action*

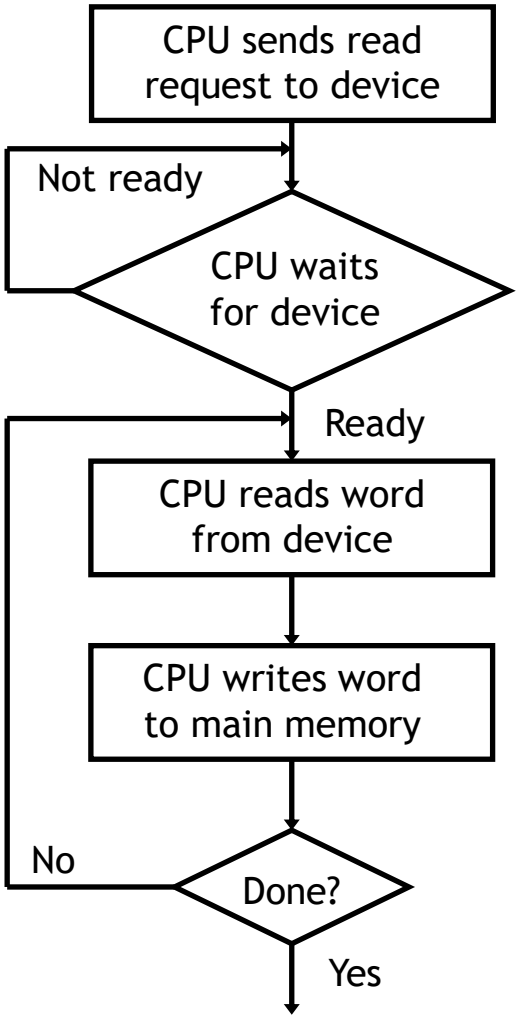
```
char read_kbd(){
  do {
    sleep();
    status = inb(0x64);
  } while(!(status & 1));

  return inb(0x60);
}
```

- Prós? Contras?
  - Temporização previsível e é barato
  - Mas: desperdiça ciclos de relógio com o CPU a não fazer nada
  - Eficiente apenas se não existe mais nada a fazer em paralelo
- Comum em sistemas embebidos ou de tempo-real

Transferência de Dados com I/O Programado

- No **I/O programado**, a transferência de dados depende de um programa do utilizador ou do sistema operativo.
- A CPU faz uma solicitação e, em seguida, espera que o dispositivo fique pronto (por exemplo, mover a cabeça de leitura de um disco).
- Os barramentos têm apenas 32-64 bits de largura, portanto, as últimas etapas são repetidas para transferências grandes.
- É necessário muito tempo de CPU para isso!
- Se o dispositivo for lento, a CPU pode ter que esperar muito tempo - a maioria dos dispositivos é lenta em comparação com as CPUs modernas.
- A CPU também está envolvida como um intermediário para a transferência de dados entre a memória e os dispositivos.



(This CPU flowchart is based on one from *Computer Organization and Architecture* by William Stallings.)

Métodos de Comunicação II

- Como é que o programa sabe se o dispositivo está pronto ou terminou a tarefa?
- Interrupções: O dispositivo envia um pedido de interrupção ao CPU
  - Existem registos específicos para identificar a causa da interrupção e o dispositivo
  - Define-se uma rotina de resposta a interrupções que decide as acções a tomar
- Esquema de prioridades
  - Eventos urgentes podem interromper o tratamento de interrupções de menor prioridade
  - O Sistema Operativo pode desactivar (ou deferir) interrupções

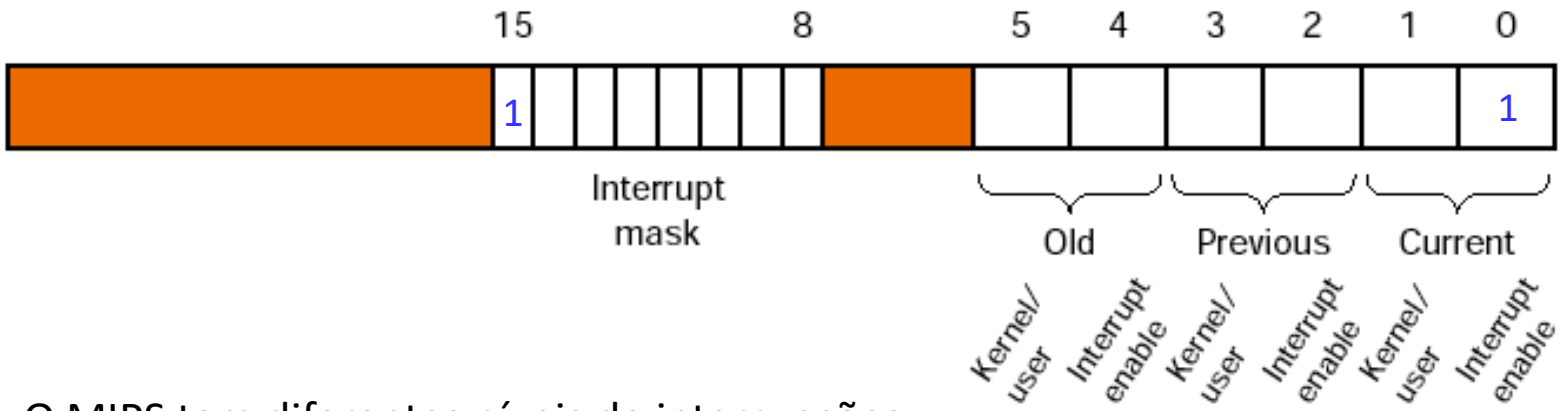
Métodos de Comunicação II

- Q: Como é que o programa sabe se o dispositivo está pronto ou terminou a tarefa?
- A: Interrupções: O dispositivo envia um pedido de interrupção ao CPU

- Prós? Contras?
  - Mais eficiente na gestão de tempo da CPU: apenas interrompe quando o dispositivo está pronto
  - Menos eficiente em termos de recursos – salvaguarda de contexto
    - Contexto do: PC, SP, registos, etc.
  - Contra: fluxo de execução de Código de temporização imprevisível uma vez que pode ser interrompido por acções externas

MIPS – Programação por Interrupções

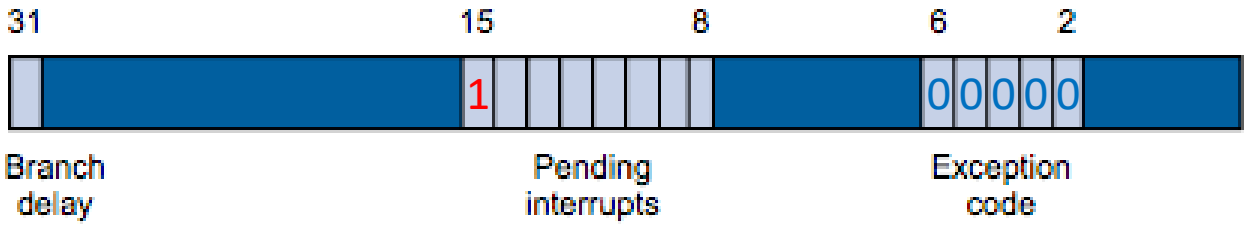
- Para receber interrupções o *software* tem de as ativar.
  - Num processador MIPS isto é feito através de uma escrita no **registo de estado**.
    - As interrupções são ativadas colocando o **bit zero** do registo de estado a 1



- O MIPS tem diferentes níveis de interrupções:
  - O mecanismo de interrupções para os diferentes níveis podem ser activados de forma selectiva.
  - Para receber uma interrupção, o bit correspondente na **máscara de interrupções** (bits 8-15 do registo de estado) deve ser colocado a 1.
    - Na figura a interrupção de nível 15 está activa.

MIPS – Programação por Interrupções

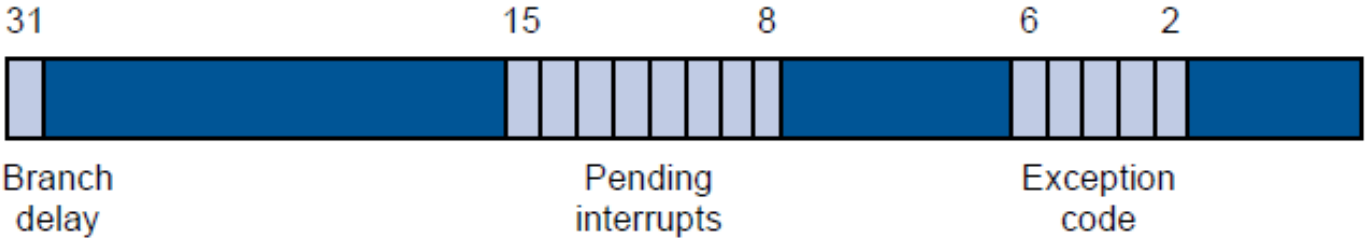
- Quando uma interrupção ocorre, o Registo de Causas (*Cause Register*) indica qual a que ocorreu.
  - Para cada excepção, o campo **exception code** guarda o tipo da excepção.
  - Quando ocorre uma **interrupção**, o campo exception code é colocado a 0000 e o bit correspondente à interrupção pendente será ativado.
    - O registo em baixo mostra uma interrupção pendente no nível 15



- A rotina de resposta às excepções é geralmente parte do sistema operativo.

Cause Register

O registo de causa contém informações sobre interrupções pendentes e os tipos de exceção que ocorrem.



Number	Name	Cause of exception
0	Int	interrupt (hardware)
4	AdEL	address error exception (load or instruction fetch)
5	AdES	address error exception (store)
6	IBE	bus error on instruction fetch
7	DBE	bus error on data load or store
8	Sys	syscall exception
9	Bp	breakpoint exception
10	RI	reserved instruction exception
11	CpU	coprocessor unimplemented
12	Ov	arithmetic overflow exception
13	Tr	trap
15	FPE	floating point

Como lidar com as Excepções?

- As **Excepções** geralmente são erros detetados no processador.
  - A CPU tenta executar uma instrução com um ***opcode*** ilegal.
  - Uma instrução aritmética causa ***overflow*** ou tenta dividir por 0.
  - Uma instrução de ***load*** ou ***store*** não pode ser realizada porque tenta aceder a uma zona de memória inválida (por alinhamento incorrecto ou por estar fora do segmento atribuído ao programa).
- Existem duas maneiras possíveis de resolver esses erros:
  - Se o erro for **irrecuperável**, o sistema operativo termina o programa.
  - Problemas menos sérios geralmente podem ser corrigidos pelo sistema operativo ou pelo próprio programa.

MIPS – Excepções

Exception Code Value		Mnemonic	Description
Decimal	Hex		
0	0x00	Int	Interrupt
1-3	0x01	—	Reserved
4	0x04	AdEL	Address error exception (load or instruction fetch)
5	0x05	AdES	Address error exception (store)
6	0x06	IBE	Bus error exception (instruction fetch)
7	0x07	DBE	Bus error exception (data reference: load or store)
8	0x08	Sys	Syscall exception
9	0x09	Bp	Breakpoint exception
10	0x0A	RI	Reserved instruction exception
11	0x0B	CPU	Coprocessor Unusable exception
12	0x0C	Ov	Arithmetic Overflow exception
13	0x0D	Tr	Trap exception
14-31	0x0E-0x1F	—	Reserved

Transferência de Dados I/O (1/4)

- Como estabelecer ligação com o dispositivo?
  - *Programmed I/O* ou *Memory-Mapped I/O*
- Como detectar eventos?
  - *Polling* ou Interrupções
- Como transferir grandes quantidades de dados?
  i. Transferência de dados Programada
  ii. Transferência utilizando Direct Memory Access (DMA)

Transferência de Dados I/O (2/4)

- **Transferência de dados Programada**:

Dispositivo ←→ CPU ←→ RAM

for (i = 1 .. n)
- CPU emite um pedido
- O Dispositivo coloca os dados no bus e a CPU lê-os para os registos
- A CPU escreve os dados na memória
- ***Não é eficiente***

Muito Lento!!

Transferência de Dados I/O (3/4)

Acesso Direto à Memória (DMA, Direct Memory Access)

1) O Sistema Operativos indica o endereço de início e o comprimento da transferência

2) O controlador (ou o dispositivo) transfere os dados para a memória autonomamente

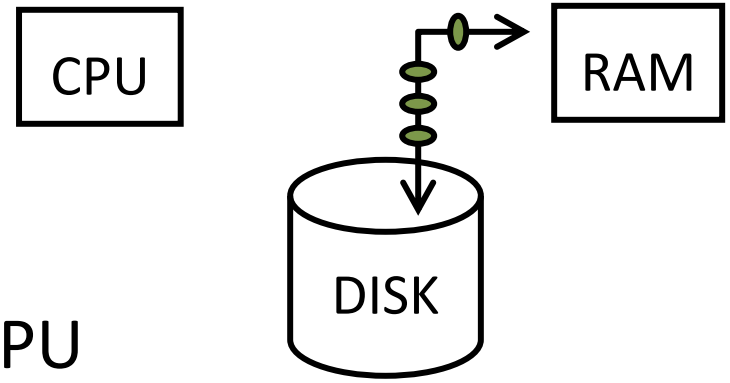
3) É gerada uma interrupção após a conclusão ou a ocorrência de um erro

Transferência de Dados I/O (4/4)

- Transferência utilizando DMA:

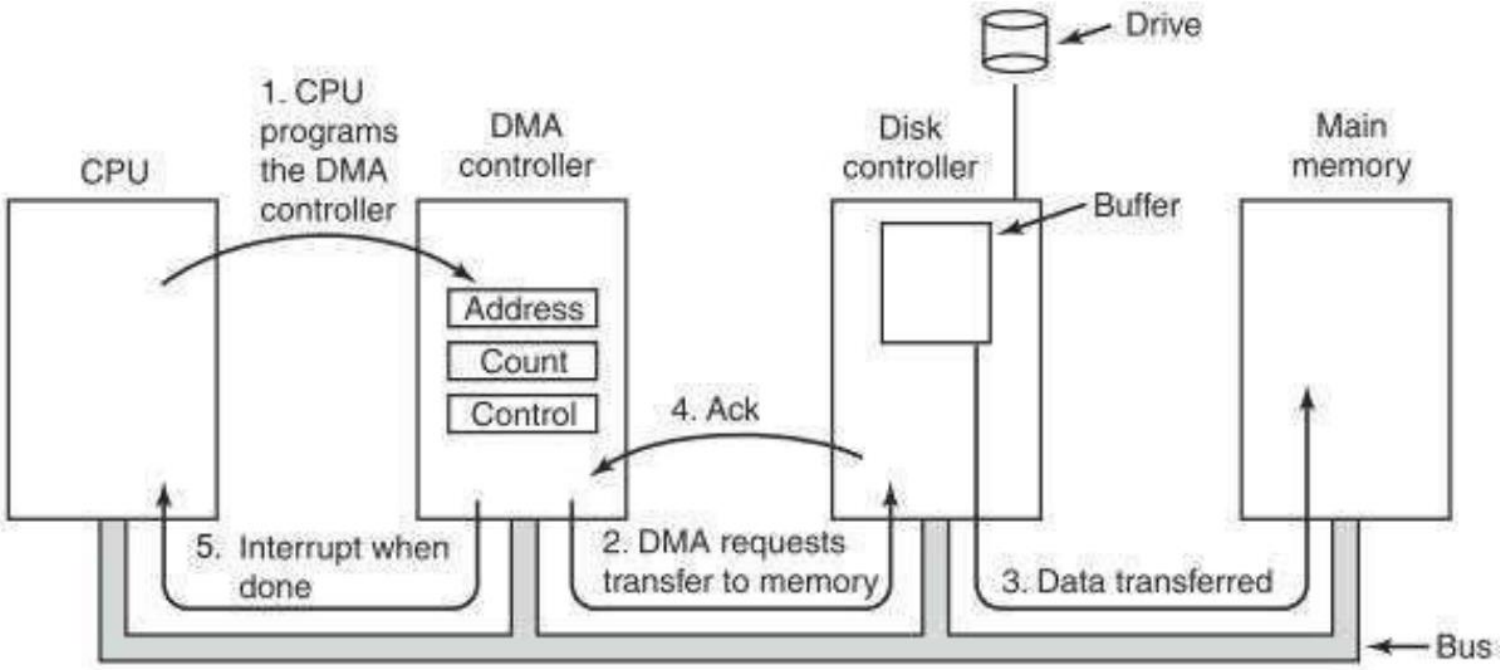
  Device ←→ RAM

  – A CPU inicia um pedido DMA
  – for (i = 1 ... n)
    O dispositivo coloca os dados no bus e a memória aceita-os
  – O dispositivo interrompe a CPU após o final da transferência ou após a ocorrência de um erro.



Exemplo

O controlador DMA solicita ao controlador do disco que transfira os dados directamente para a memória.



Para saber mais ...

- P&H - Capítulos 4.5 a 4.8 inclusive