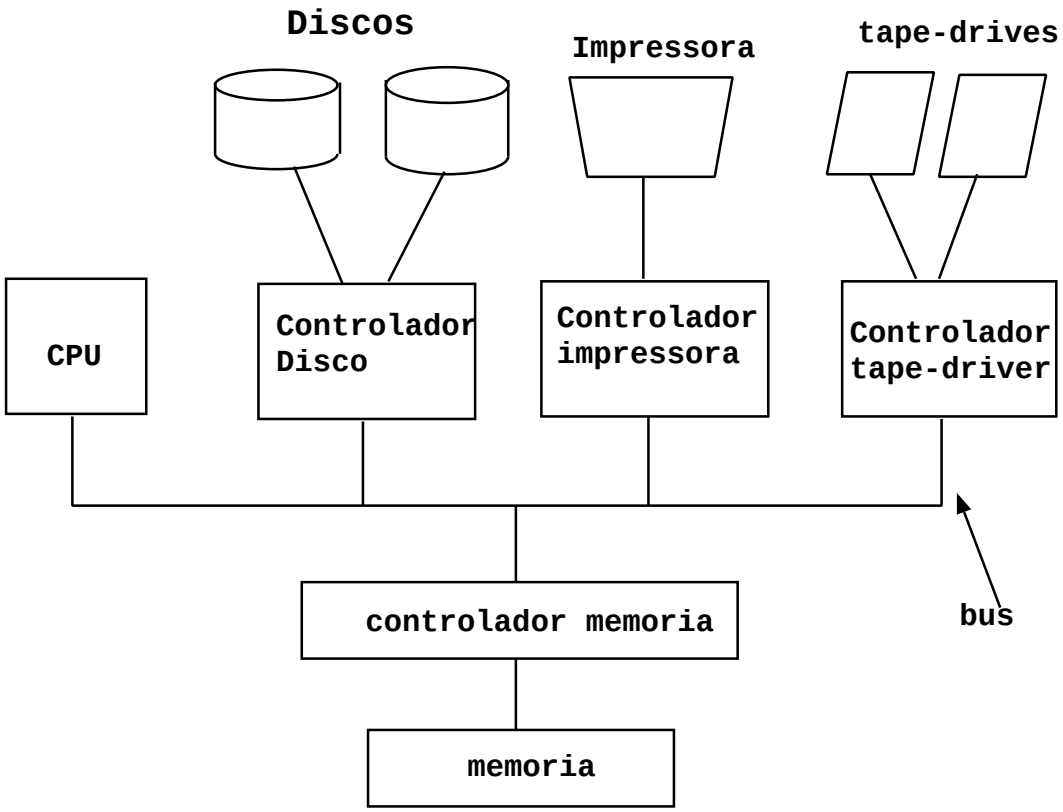
Arquitetura de um Computador

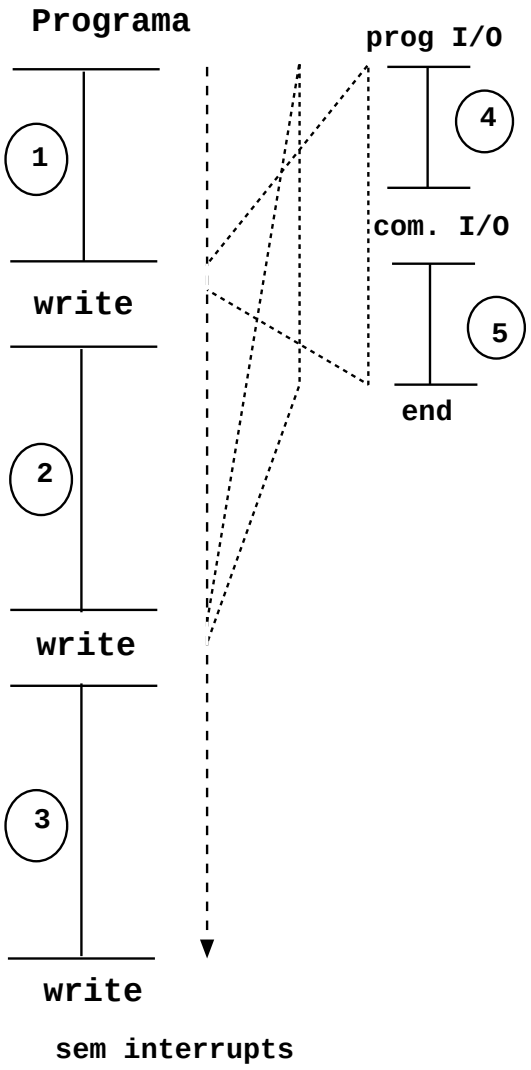


- Os periféricos de I/O e o CPU executam concorrentemente.

- Cada controlador de periférico encarrega-se de um tipo particular de periférico.
- Cada controlador de periférico tem um buffer local.
- O CPU desloca dados de/para a memória principal para/de buffers locais.
- I/O acontece do periférico para o buffer local do controlador.
- O controlador do periférico informa o CPU que terminou a sua operação, causando uma interrupção (*interrupt*).

I/O sem Interrupções

Na figura seguinte, os segmentos 1, 2 e 3 referem-se a sequências de instruções que não envolvem I/O. As chamadas a `write` destinam-se a um programa de I/O, que é um utilitário do sistema, que realiza a operação de I/O.



O programa de I/O envolve:

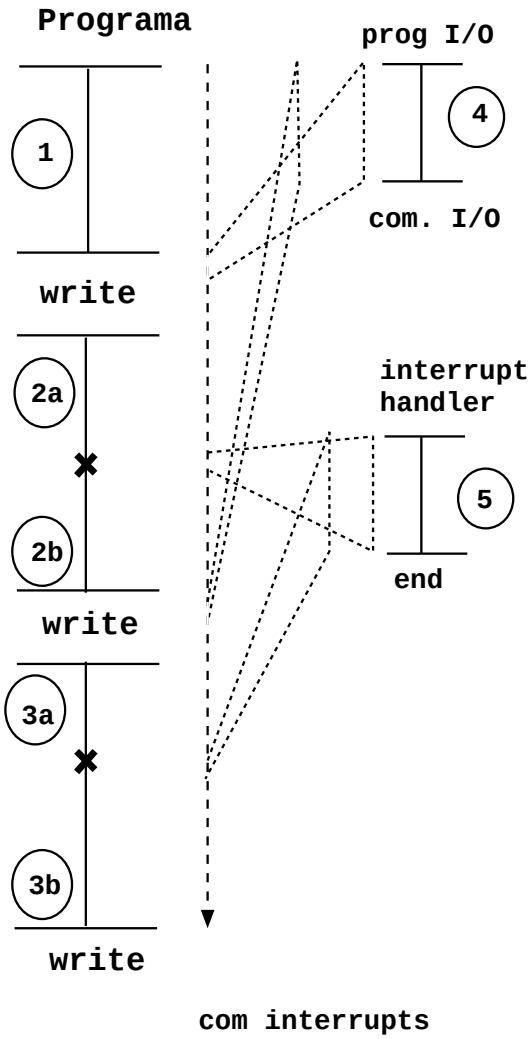
- parte 4, instruções que preparam a operação. Pode envolver cópia de dados para um buffer e faz a chamada do comando de acesso ao periférico.
- O comando propriamente dito. Sem interrupts, quando este comando é emitido a CPU fica à espera que o periférico realize a operação.
- parte 5, instruções que completam a operação. Pode envolver mudar um “flag” para indicar sucesso ou falha na operação.

Como a operação de I/O pode tomar muito tempo, o programa de I/O fica à espera que termine e portanto o programa do utilizador fica parado na instrução `write`.

O programa de I/O corresponde ao device-driver.

Interrupções (Interrupts)

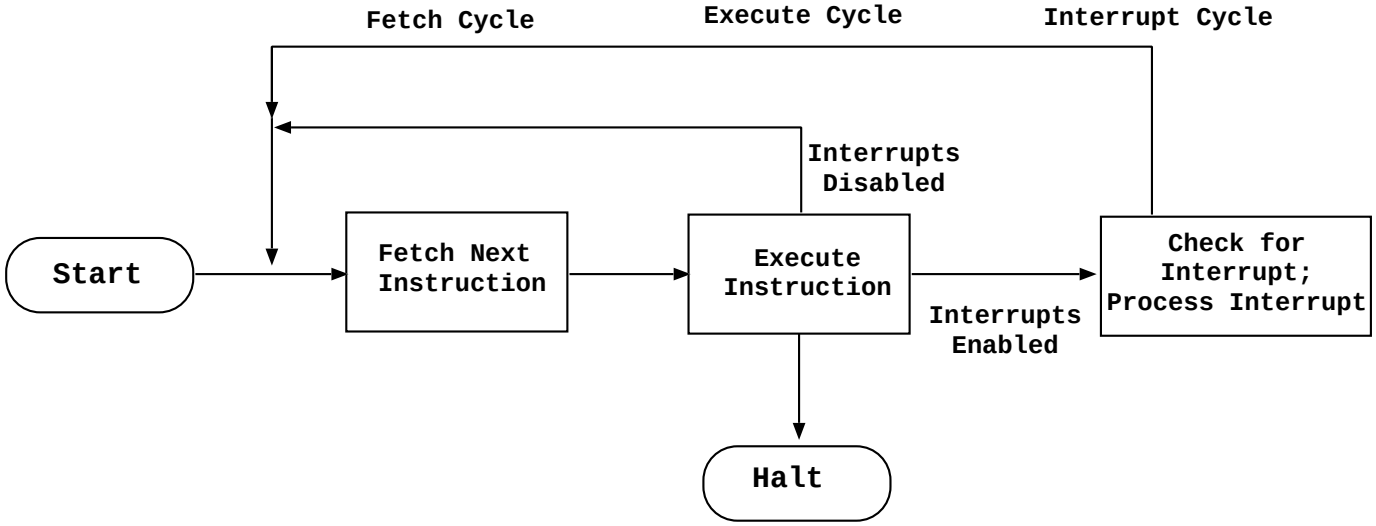
- Servem essencialmente para melhorar a eficiência de processamento.
- Permitem que o CPU execute outras instruções enquanto a operação de I/O se realiza.



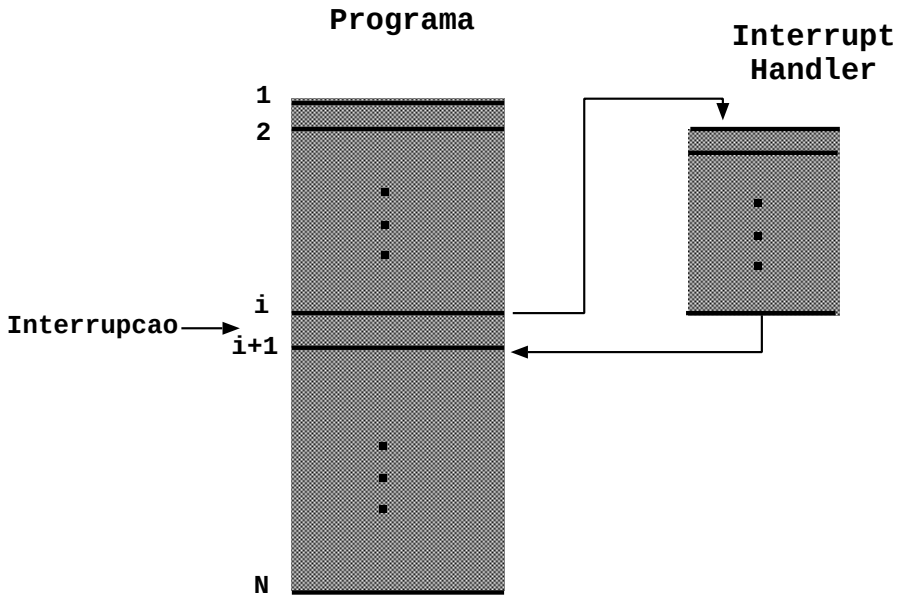
- com `write`, o programa de I/O é chamado para preparar o código de chamada do comando de acesso ao periférico. O controlo retorna ao programa do utilizador.
- Quando a operação de I/O tiver sido realizada, este módulo envia um sinal de interrupt ao CPU.
- O CPU suspende a execução do programa corrente, passa a execução para o *interrupt handler* e retoma o programa inicial quando este periférico terminar.

O interrupt-cycle

Para lidar com interrupts é necessário adicionar um interrupt-cycle ao ciclo de instruções habitual.



O *Interrupt handler* é parte do SO. Determina a natureza do interrupt e, por exemplo, chama o programa que acede ao periférico. Quando termina, o CPU executa outro processo do sistema ou retoma a execução do programa.


Programa
Interrupt
Handler
1
2
i
Interrupcao
i+1
N

O SO e interrupções

Um SO é conduzido por interrupções:

- quando ocorre uma interrupção (ou trap), o hardware transfere o controlo para o SO.

- Uma armadilha (trap) é uma interrupção gerada por software e que pode surgir devido a um erro ou a pedido do utilizador.

- o SO, primeiro preserva o estado do CPU, guardando os registos e program-counter.

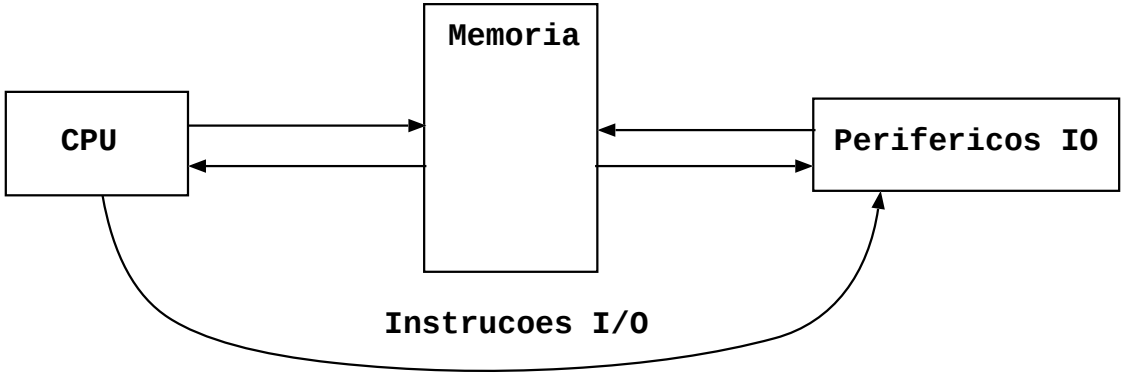
- depois determina qual o tipo de interrupção que ocorreu – o que pode ser feito por *polling* dos periféricos, ou pode ser resultado do vetor de interrupções do sistema.

- para cada tipo de interrupção, diferentes segmentos de código determinam a acção a tomar.

- quando se inicia a operação de I/O, o controlo retorna para o SO que poderá

escolher outro processo para aceder ao CPU.

DMA: Direct Memory Access



- é usada para periféricos I/O de alta velocidade, capazes de transmitir informação a velocidade próxima da velocidade da memória.
- o controlador do periférico transfere blocos de dados do buffer directamente para a memória sem intervenção do CPU.
- apenas gera uma interrupção por bloco, em vez de uma por cada byte!.

Velocidade de periféricos

Item	Tempo		Escala do Tempo Humano (100 milhões vezes + devagar)	
ciclo CPU	10 ns	(100MHz)	1	segundo
acesso a cache	30 ns		3	segundos
Acesso a memória	60 ns		6	segundos
Troca contexto	10,000 ns	(100 μs)	166	minutos
Acesso a disco	10,000,000 ns	(10 ms)	11	dias
Quantum	100,000,000 ns	(100 ms)	116	dias

Protecção a nível do hardware

- Dois modos de operação
- Protecção de I/O
- Protecção de acesso a memória
- Protecção do CPU

Dois Modos de Operação (Dual Mode)

- a partilha de recursos, requer do SO a certeza que nenhum programa incorreto pode ocasionar o incorreto funcionamento de outros.
- fornece suporte de hardware para diferenciar entre dois modos de operação:
  - Modo do utilizador – execução feita ao serviço do utilizador.
  - Modo de kernel (ou supervisor) – execução feita pelo SO.
- *bit-mode* – adicionado ao hardware para indicar o modo corrente: kernel(0) ou utilizador(1).
- Quando uma interrupção ou falha ocorre, o hardware passa para o modo de kernel.

Protecção de I/O

- Tem de assegurar que o programa não vai nunca ganhar controlo do computador, ou realizar I/O indevido.
- Solução: as operações de I/O são realizadas em modo kernel.
- i.e. os utilizadores não acedem directamente aos periféricos!

Protecção de acesso a memória

• deve fornecer protecção pelo menos no acesso ao vetor de interrupções.

• usar dois registos que determinam o alcance dos endereços que um programa pode aceder.

- registo base (RB) – contém o menor endereço.
- registo limit (RL) – contém o tamanho da variação.

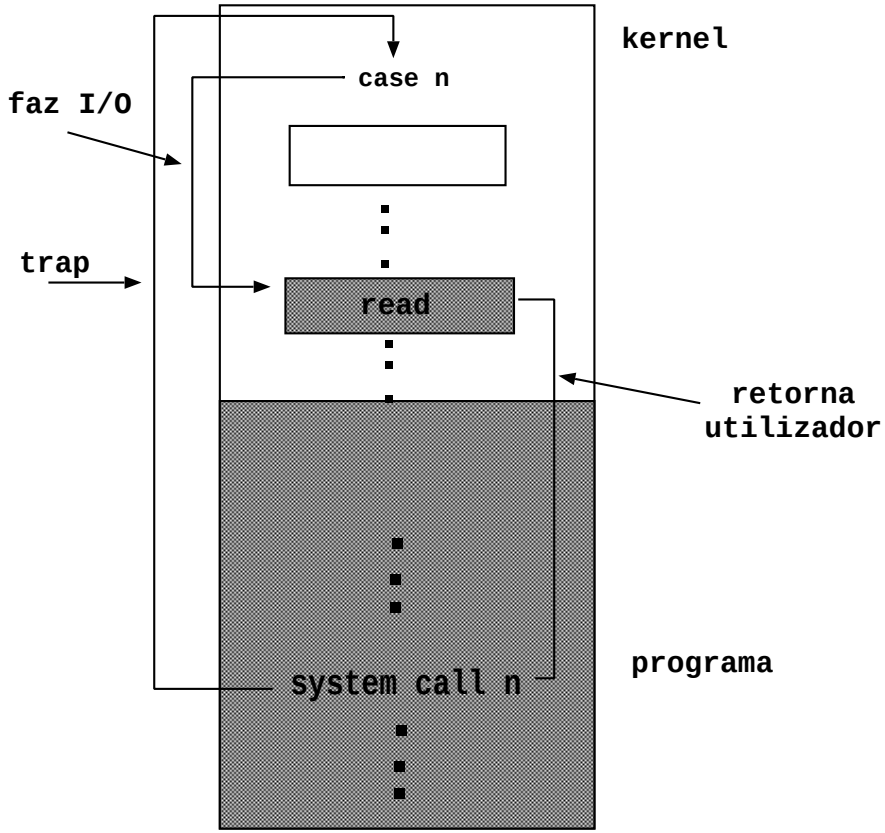
Deste modo acessos a endereços x tal que $x < RB \; ou \; x > (RB + RL)$ originam um "trap" correspondente a *segmentation-fault*.

Protecção do CPU

- É preciso evitar que caso um programa do utilizador entre em deadlock (um ciclo infinito), o SO perca controlo e fique inoperacional.
- Temporizador (Timer) – interrompe o computador depois de decorrido um certo tempo, permitindo que o SO tome controlo.
  1. é inicializado com um certo número de ticks.
  2. é decrementado em cada tick do relógio da máquina.
  3. quando chega a 0, gera um interrupt, e volta a 1.
- Timer – usado para suportar time-sharing.
- Timer – usado também para calcular a hora actual.
- A inicialização do Timer faz-se em modo kernel (protegido).

Então como usar o sistema?

- Dado que as instruções de I/O são protegidas (são executadas em modo kernel), como é que um programa do utilizador faz I/O?
- Chamadas de sistema – o método usado por um processo para pedir a intervenção do SO:
  - toma habitualmente a forma de uma armadilha (trap) para um endereço no vetor de interrupções.
  - o controlo passa para o interrupt-handler e o mode-bit é colocado em modo kernel.
  - o kernel verifica se os parâmetros estão corretos, executa o pedido e devolve o controlo para a instrução seguinte à chamada ao sistema.


kernel
case n
faz I/O
trap
read
retorna
utilizador
programa
system call n