Entrada/Saída

Entre outras funções, o SO também controla os dispositivos de E/S:

- emite comandos para dispositivos
- intercepta interrupções
- trata os erros
- fornece interface entre os dispositivos de E/S e o resto do sistema

Princípios do HW de E/S

Hardware pode ser visto de 2 formas:

- ✓ Engenheiro eletrônico: componentes (chips), cabos, suprimento de energia, entre outros;
- ✓ Programadores: interface apresentada ao SW (comandos que o HW aceita, funções que realiza, além de erros repassados ao SW)

Nosso interesse: programação dos dispositivos de E/S (apesar de estarem ligadas às operações internas)

Dispositivos de E/S

Podem ser divididos em duas categorias:

Dispositivos de blocos:

Armazenam as informações em blocos de tamanho fixo, cada um com seu próprio endereço

Ex: *HD, Pendrive, Blu-ray*

Dispositivos de caracteres:

Enviam ou recebem um fluxo de caracteres sem considerar qualquer estrutura de blocos

Ex: *impressoras, placas de rede, mouses, teclados*

Dispositivos de E/S

Observações:

1. Alguns dispositivos não se encaixam em nenhuma das duas classificações. Ex: ***Relógios, telas mapeados na memória, telas de toque***

2. Os sistemas de arquivos lidam com dispositivos de blocos abstratos. O tratamento da parte dependente de dispositivos é função do SW de baixo nível

Dispositivos de E/S

Características de dispositivos de blocos:

→ Tamanhos dos blocos variam entre 512 bytes e 64KB

→ Transferências em unidades inteiras de blocos

Propriedade essencial:

cada bloco pode ser lido ou escrito independentemente de todos os outros

Dispositivos de E/S

Características de dispositivos de caracteres

→ Envia e recebe um fluxo de caracteres sem considerar qualquer estrutura de blocos

→ Não são endereçáveis

→ Não dispõem de operações de posicionamento

Controladores de Dispositivos de E/S

Unidades de E/S consistem de:

- **Componente mecânico:** dispositivo propriamente dito

- **Componente eletrônico:** controlador de E/S (em PCs, chips ou placas de circuito impresso)

Controladores de Dispositivos de E/S

Controlador de E/S:

- Conector + cabo que o conecta ao dispositivo
- 1 controlador pode tratar 2, 4 ou mais dispositivos idênticos
- **Interfaces padrões (entre o controlador e o dispositivo)**
  - ✓ IDE, SATA, SCSI, USB

    ⇒ facilitam o desenvolvimento dos controladores pelas empresas

Controladores de Dispositivos de E/S

- Interface entre o controlador e o dispositivo é de baixo nível (bits)

- Principal função do controlador de E/S:

Converter bits seriais em dados/blocos inteligíveis pelo SO

Para isso:

monta os bits em um buffer, faz a correção de eventuais erros e copia para a memória

Controladores de Dispositivos de E/S

Funcionamento:

Exemplo 1: **Disco formatado com 10 mil setores de 512 bytes por trilha.**

- Unidade de disco entrega ao controlador um fluxo serial de bits

  = preâmbulo (cilindro, setor e tamanho do setor) + 4096 bits/setor + checksum, etc

- Controlador converte fluxo de bits em um bloco de bytes
- Monta os blocos em um buffer
- Executa a correção de erros
- Copia bloco para a memória.

Controladores de Dispositivos de E/S

Exemplo 2: **Controlador de um monitor de vídeo**

- Lê bytes da memória que contêm caracteres para serem mostrados na tela

- Gera os sinais eletrônicos que fazem com que os caracteres sejam escritos na tela

  $\Rightarrow$ facilita a vida do programador do sistema operacional, que não precisa se preocupar em programar em baixo nível

E/S mapeada na memória

Nos computadores mais antigos (IBM-360 e sucessores) a comunicação entre o controlador dos dispositivos de E/S e a CPU era realizada da seguinte forma:

- Controlador tem registradores de controle ou buffer de dados (Ex.: RAM de video) para a comunicação com a CPU
- Através desses registradores o SO comanda os dispositivos para entregar ou aceitar dados, ligar e desligar esses dispositivos, etc
- SO também descobre o estado do dispositivo, se está preparado para aceitar dados, etc

E/S mapeada na memória

Nesta abordagem:

- Cada registrador de controle é associado a um número de **porta de E/S**
- Conjunto das portas forma o **espaço de portas de E/S (somente o SO pode acessá-las)**

Exemplos: **IN REG, PORT** (lê o valor do registrador de controle **PORT** no registrador de uso geral da CPU);

OUT PORT, REG (escreve o conteúdo de **REG** no registrador de controle)

⇒ ***Espaços de endereçamento para a memória e para as portas de E/S são separados (fig 5.1a):***

IN R0, 4 ≠ MOV R0, 4

E/S mapeada na memória

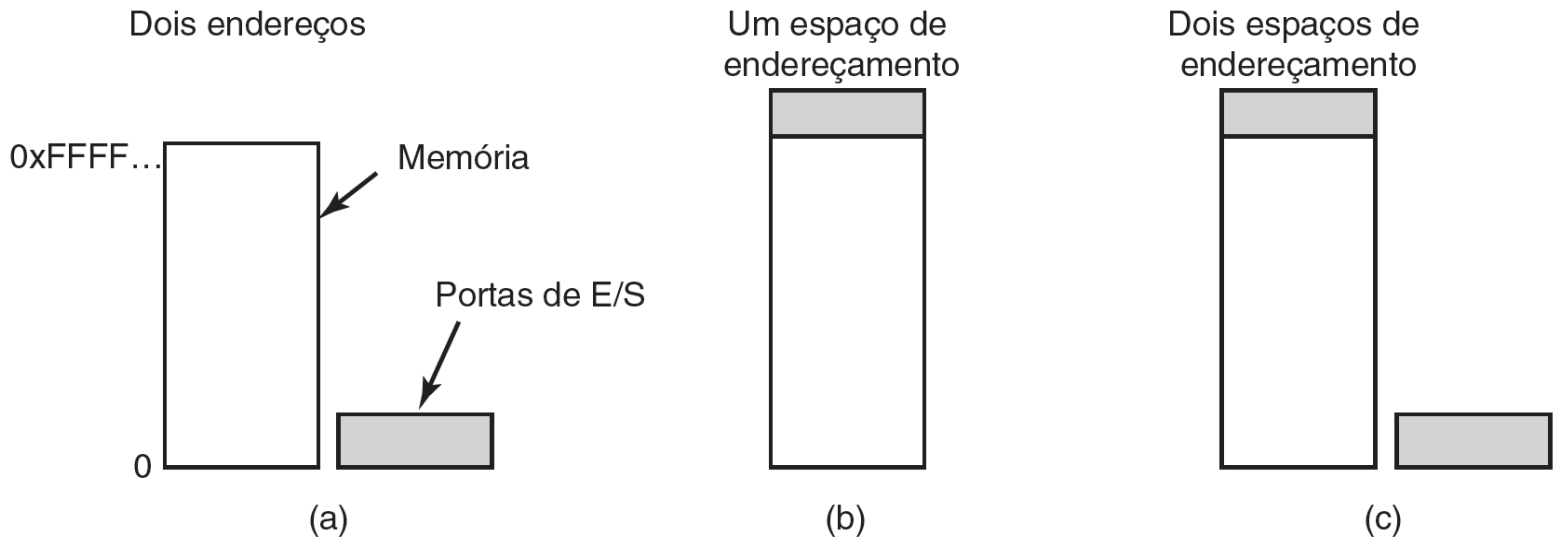

Figura 5.1 (a) Espaços de memória e E/S separados. (b) E/S mapeada na memória. (c) Híbrido.

E/S mapeada na memória

Alternativa para integrar os dois espaços de endereçamento:

- Usada no PDP-11: mapeia todos os registradores de controle no espaço de endereçamento da memória (fig 5.1b)

  ⇒ ***E/S mapeada na memória***

- Cada registrador de controle (porta de E/S) é associado a um endereço de memória único

- Endereços associados ficam localizados no topo do espaço de endereçamento

Híbrido

Outra abordagem:

- Utilizada na arquitetura x86
- Endereços de 640K a 1M – 1 $\Rightarrow$ reservados para buffer de dados dos dispositivos
- Portas de E/S de 0 a 64K – 1 $\Rightarrow$ em espaço de end. diferente
- Possui uma linha de barramento adicional para indicar se quer ler/escrever de/em um espaço de endereçamento ou de/em outro

Híbrido

Funcionamento:

Ex.: CPU quer ler uma palavra da memória ou de uma porta de E/S

– Coloca o endereço no barramento

– Emite sinal READ sobre uma linha de controle

– Linha adicional informa se é espaço de E/S ou memória

– Memória ou dispositivo de E/S responde à requisição

E/S Mapeada na Memória

Funcionamento:

- Compara-se as linhas de endereço com a faixa associada a cada um (memória ou portas de E/S)
- Se endereços estão na faixa de um determinado componente, este responde à requisição
- Não há ambiguidade ou conflito, pois o endereço está associado somente às portas de E/S ou à memória

Vantagens da E/S mapeada na memória

- Portas são variáveis comuns

⇒ **Driver do dispositivo de E/S pode ser escrito em C:** não é necessário um outro código em assembly para usar IN/OUT ⇒ **causaria um custo adicional**

- Não é preciso mecanismo de proteção especial para processos do usuário porque o ***espaço de endereçamento virtual do usuário fica limitado à porção da memória permitida***

- Pode-se colocar os registradores de controle de cada dispositivo numa página diferente

⇒ ***SO pode dar privilégios a alguns usuários incluindo ou não as páginas em sua tabela de páginas***

Vantagens da E/S mapeada na memória

- Como diferentes drivers de dispositivos podem ser colocados em diferentes espaços de endereçamento

  ⇒ ***evita que um problema em um driver interfira em outro***

- Cada instrução capaz de referenciar a memória também pode referenciar os registradores de controle para realizar testes nos dispositivos de E/S

```
Ex:      LOOP:  TEST port_4  // verifica se porta 4 é 0 ⇒ disp. ocioso
                BEQ   READY // se for, salta para ready
                BRANCH LOOP  // senão, continua testando
         READY:
```

Obs: **Se não existir E/S mapeada na memória, é necessária 1 instrução a mais (reg de controle deve primeiro ser lido para a um registrador de uso geral da CPU e depois testado, atrasando o teste de ociosidade)**

Desvantagens da E/S mapeada na memória

- Uso de cache no exemplo anterior: futuras referências à cache usariam valor antigo de PORT_4 ⇒ loop infinito, programa não saberia qdo o dispositivo estivesse pronto

  Solução: HW deve desabilitar a cache

  ⇒ aumenta a complexidade do HW e do SO

- Com apenas 1 barramento (fig 5.2a), não teria problema

- fig 5.2b – Com dois barramentos (para aumentar desempenho), os dispositivos de E/S não têm como enxergar os endereços de memória (lançados no barramento de memória) e não têm como responder ⇒ mais complexidade no HW

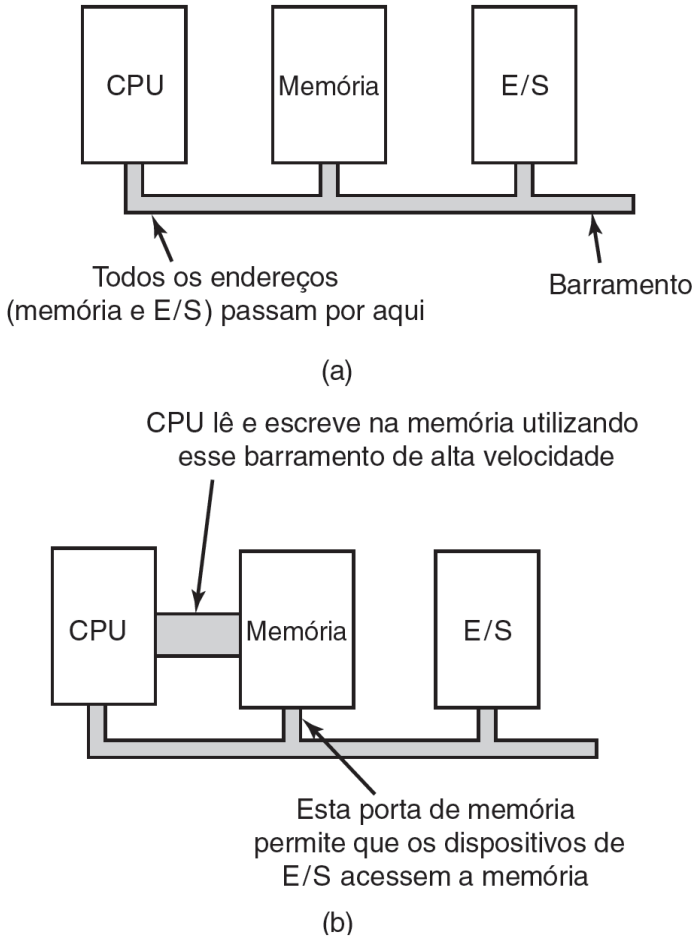

Figura 5.2 (a) Arquitetura com barramento único. (b) Arquitetura de memória com barramento duplo.

Soluções p/ as desvantagens da E/S mapeada na memória

Solução 1: se o 1º acesso falhar, CPU testa outros barramentos

Solução 2: colocar um dispositivo de escuta no barramento memória para repassar aos dispositivos de E/S os endereços apresentados.

Problema: ***dispositivos de E/S podem não ser capazes de processar os pedidos na mesma velocidade da memória***

Solução 3 (x86): filtrar os endereços reservados nos registradores do controlador de memória durante a inicialização do SO

⇒ endereços na faixa de 640K a 1M – 1, por exemplo, são transferidos para o barramento PCI ao invés de serem transferidos para a memória.

Acesso direto à memória (DMA)

Motivação:

CPU, normalmente, precisa endereçar os controladores para trocar dados com eles ⇒ desperdiça muito tempo de CPU

Controlador de DMA:

libera a CPU para tarefas mais importantes e fica responsável por acessar os controladores de E/S

Tipos:

- localizado no próprio dispositivo (nesse caso, seria necessário um DMA para cada dispositivo diferente)
- separado (geral para todos os dispositivos), localizado na placa-mãe (solução mais freqüente)

Acesso direto à memória (DMA)

Características:

- acesso ao barramento de forma independente da CPU
- vários registradores, que podem ser lidos e escritos pela CPU:
  - ✓ **registrador de endereçamento de memória**
  - ✓ **registrador contador de bytes**
  - ✓ **um ou mais registradores de controle, que especificam:**
    - **porta em uso**
    - **direção da transferência de dados (L/E de/para disp E/S)**
    - **unidade de transferência (byte ou palavra)**
    - **nº de bytes a ser transferido em um surto**

Acesso direto à memória (DMA)

Leitura sem DMA:

1. Controlador de disco lê um bloco do dispositivo serialmente (bit a bit) até que todo o bloco esteja no buffer interno;

2. Calcula Checksum (para verificar se houve erro de leitura);

3. Controlador de disco gera uma interrupção;

4. SO lê o bloco gravado no buffer do controlador. A cada iteração de um loop, lê um byte ou uma palavra de cada vez de um registrador do controlador e armazena-a na memória;

Acesso direto à memória (DMA)

Leitura com DMA:

1. CPU programa o controlador de DMA configurando seus registradores (o que e para onde transferir)

2. CPU emite comando para o controlador de disco para carregar os dados do disco para o buffer interno do mesmo e verificar o *checksum*. CPU é liberada e o DMA inicia seu trabalho.

3. DMA emite uma requisição de leitura, via barramento, para o controlador de disco (este sabe onde escrever a palavra na memória através das linhas de endereço no barramento)

4. Controlador de disco começa a transferência dos dados para a memória

Acesso direto à memória (DMA)

5. Quado termina, controlador de disco envia sinal de confirmação (via barramento) ao controlador de DMA

6. Passos 2 a 5 são repetidos até que o contador de bytes/palavras (do DMA) a serem transferidos chegue a zero

7. Quando contador = 0, DMA interrompe CPU para avisar do fim da transferência de dados para a memória

Acesso direto à memória (DMA)

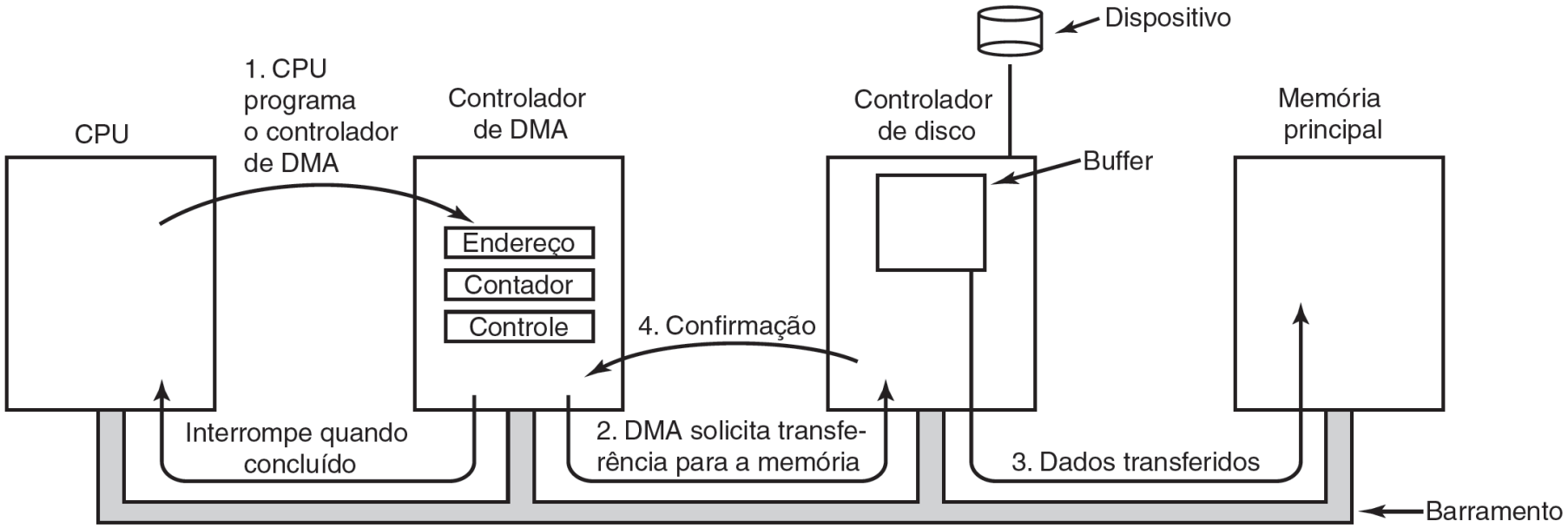

Figura 5.3 Operação de transferência utilizando DMA.

Acesso direto à memória (DMA)

Observações:

1. Podem existir vários conjuntos de registradores no DMA, um para cada dispositivo;
2. Após a transferência de cada palavra o DMA decide qual dispositivo será o próximo a ser atendido: esquema usado pode ser por prioridades ou ***round-robin***
3. Operação do DMA pode ser:
   - em **modo palavra** (transferência palavra por palavra, roubando ciclos da CPU caso a mesma também precise do barramento)
   - em **modo surto** (transferência de várias palavras em uma única aquisição de barramento). Mais eficiente, mas pode bloquear a CPU por mais tempo, caso o surto seja longo

Acesso direto à memória (DMA)

Observações:

4. **modo direto** – dispositivo transfere os dados diretamente para a memória

   modo indireto – **dispositivo → DMA → memória**

   Requer mais uma solicitação do barramento, porém é mais flexível, pois permite cópia de dispositivo p/ dispositivo ou de memória p/ memória.

5. Normalmente, o DMA usa endereçamento físico (SO converte endereço virtual para físico e escreve no registrador de endereços do DMA).

   Outra opção é o DMA usar endereçamento virtual (DMA usa a MMU para converter para endereço físico). Isto só é possível se a MMU faz parte da memória (raro, mas possível).

Acesso direto à memória (DMA)

Observações:

6. Um buffer interno no controlador do disco é necessário para ser realizado checksum e para o caso do barramento estar ocupado

7. Sistemas embarcados (baixo custo) não usam DMA

8. Alguns projetos de computadores argumentam que a CPU é muito mais rápida e esperar pelo DMA terminar não faz sentido, caso a CPU esteja ociosa

Interrupções

Como ocorrem e são tratadas:

a) Solicitação

- ✓ dispositivo de E/S finaliza seu trabalho;
- ✓ gera um sinal de interrupção pela linha do barramento ao qual está associado (desde que estejam habilitadas pelo SO);
- ✓ sinal é detectado pelo controlador de interrupções, que fica na placa-mãe;
- ✓ se nenhuma outra interrupção está pendente, controlador processa a interrupção;
- ✓ se outra interrupção estiver em andamento ou alguma de maior prioridade tiver sido solicitada, o pedido é ignorado e o dispositivo continua a gerar o sinal no barramento até ser atendido;

Interrupções

b) Tratamento:

- ✓ controlador de interrupções coloca um número nas linhas de endereço para identificar dispositivo e repassa um sinal para interromper CPU;
- ✓ CPU interrompe o que está fazendo;
- ✓ nº das linhas de endereço é usado como índice de uma tabela chamada ***vetor de interruções*** para buscar um novo valor para o PC;
- ✓ PC agora aponta para o início de uma ***rotina de tratamento de interrupções*** correspondente ao dispositivo;

Interrupções

- ✓ a rotina de tratamento interrupção pode estar em HW ou em algum lugar da memória. Neste caso, um registrador da CPU, carregado pelo SO, aponta para a origem da rotina;

- ✓ a rotina de tratamento da interrupção confirma a interrupção, escrevendo um valor em uma das portas de E/S do controlador de interrupções;

- ✓ a confirmação indica ao controlador que ele está livre para repassar outra interrupção;

Esse pequeno atraso até a confirmação evita condições de disputa envolvendo múltiplas (quase simultâneas) interrupções

Interrupções

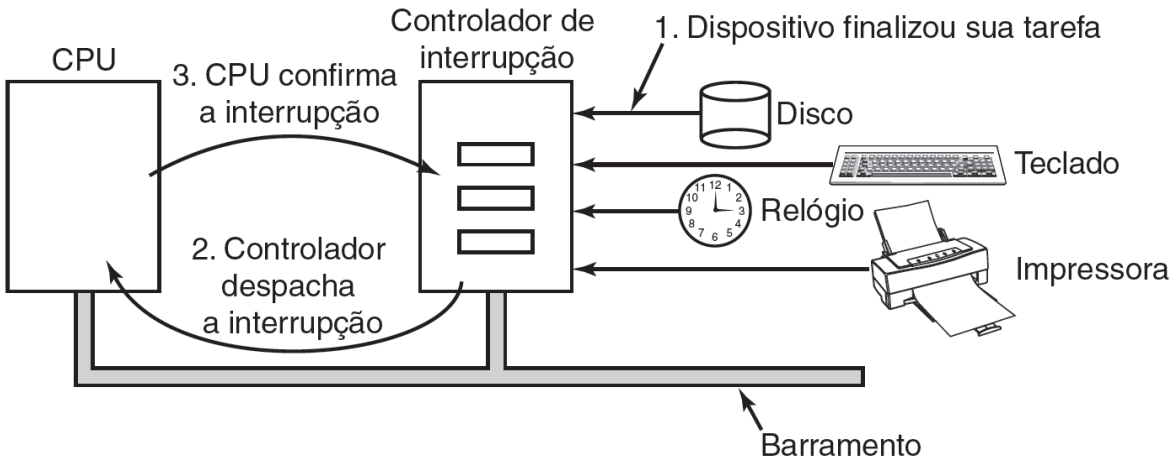

Figura 5.4 Como ocorre uma interrupção. As conexões entre os dispositivos e o controlador de interrupção atualmente utilizam linhas de interrupção no barramento, em vez de cabos dedicados.

Interrupções

Observações:

1. O HW deve “salvar o contexto” do programa em execução antes de tratar a interrupção;

2. Enquanto esse processo é feito o controlador de interrupções não pode receber a confirmação da interrupção (p/ evitar que uma segunda solicitação sobreponha os valores dos registradores da primeira enquanto esta ainda estiver sendo completada)

3. São usadas pilhas para salvar as informações do programa. A recarga dessas informações torna o processamento de interrupções lento e desperdiça tempo de CPU

Interrupções precisas e imprecisas

Motivação:

Em máquinas antigas

- **Após o fim da execução de uma instrução, verificava-se se havia uma interrupção pendente;**
- **Em caso afirmativo, PC e PSW eram salvos na pilha e a sequência de interrupção começava**
- **Após o tratamento da interrupção era realizada a sequência reversa e o processo reiniciado**

Interrupções precisas e imprecisas

Em máquinas com pipelines longos ou superescalares

- vários estágios e vários pipes

  $\Rightarrow$ conceito de instrução corrente é nebuloso!!

- PC apenas indica a próxima instrução a ser buscada na memória e não a que acabou de executar

- execução das instruções não é sequencial: cada uma pode estar em um estágio diferente, dependendo dos recursos disponíveis

Interrupção precisa **é uma interrupção que deixa a máquina em um estado bem definido quando for atender à mesma**

Interrupções precisas e imprecisas

Propriedades de uma *interrupção precisa*:

1) O PC é salvo em local conhecido.

2) Todas as instruções anteriores àquela apontada pelo PC foram executadas.

3) Nenhuma instrução posterior à apontada pelo PC foi concluída (mas podem ser iniciadas e retrocedidas para desfazer seus efeitos)

4) O estado de execução da instrução apontada pelo PC é conhecido.

Interrupções precisas e imprecisas

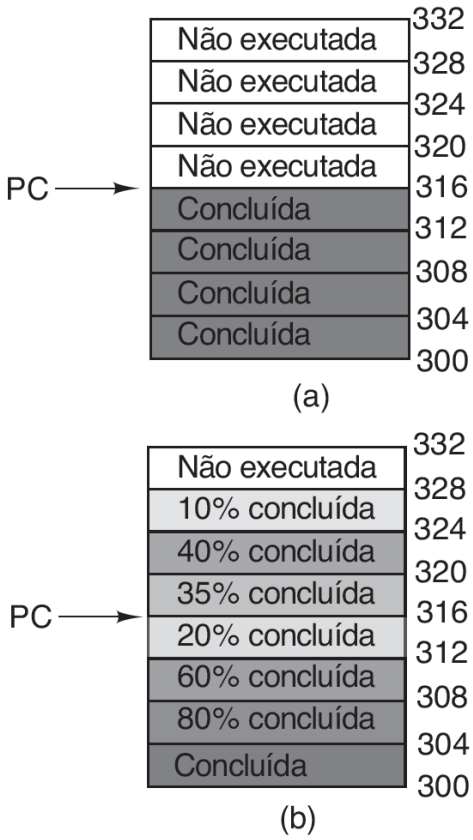

Figura 5.5 (a) Uma interrupção precisa. (b) Uma interrupção imprecisa.

Interrupções imprecisas

Características:

- Máquinas com interrupções imprecisas colocam grande quantidade de info de estado interno em uma pilha, possibilitando ao SO saber o que estava acontecendo antes da interrupção
- Código para reiniciar a máquina é complicado
- Interrupção/Recuperação é lenta devido à grande quantidade de informações que se deve salvar na memória

Problema:

Lentidão das interrupções ⇒ CPUs rápidas (superescalares) podem ser inadequadas para o trabalho em tempo real

Interrupções imprecisas

Observações:

1. Alguns computadores têm interrupções precisas (causadas por dispositivos de E/S) e outras imprecisas (causadas por erros fatais);

2. Outros têm um bit que pode forçar as interrupções a serem precisas $\Rightarrow$ sobrecarga de trabalho para a CPU $\Rightarrow$ queda no desempenho

3. Alguns computadores superescalares (família x86) têm interrupções precisas para permitir que softwares antigos rodem corretamente. Custo: área do chip (complexidade do projeto)

Dilema: maior cache ou interrupções precisas?