EA075
Comunicações: Interface e Barramentos

Faculdade de Engenharia Elétrica e de Computação (FEEC)
Universidade Estadual de Campinas (UNICAMP)

Prof. Rafael Ferrari

(Documento baseado nas notas de aula do Prof. Levy Boccato)

Introdução

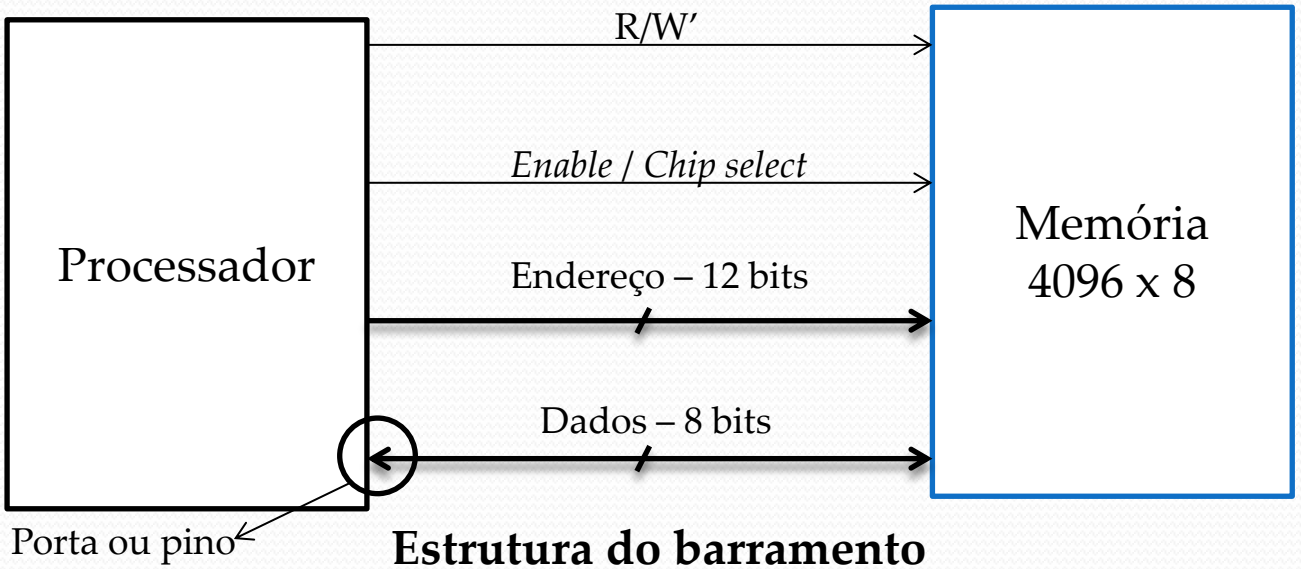
- Outro aspecto fundamental no projeto de sistemas embarcados está ligado à elaboração de um mecanismo de **comunicação** eficiente entre os dispositivos.

- O caso mais comum refere-se à conexão entre o processador e a memória visando a transferência de dados.

- Implementação: **barramentos**.

Introdução

- **Barramento:** conjunto de fios (conexões) dedicados a uma função.
  - Por exemplo, barramento de endereço, barramento de dados, etc.
  - Também se usa o termo barramento para denominar uma coleção completa de fios (e.g., endereço + dados + controle).



Estrutura do barramento

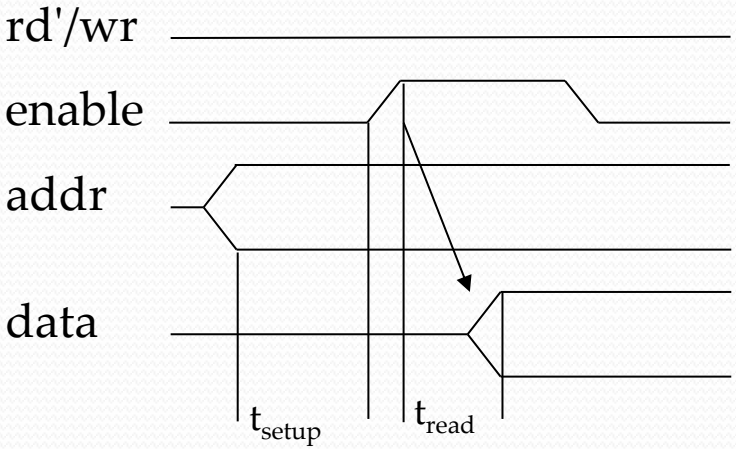
Introdução

- **Protocolo:** conjunto de regras que determina o *modus operandi* de um barramento.
  - Um protocolo estabelece mecanismos para que a comunicação entre transmissor e receptor seja bem-sucedida.
  - Um protocolo pode ser composto por subprotocolos, tais como protocolo de leitura e de escrita.
  - Cada subprotocolo é chamado de transação ou ciclo do barramento.
  - Um ciclo do barramento pode consumir vários ciclos de relógio.

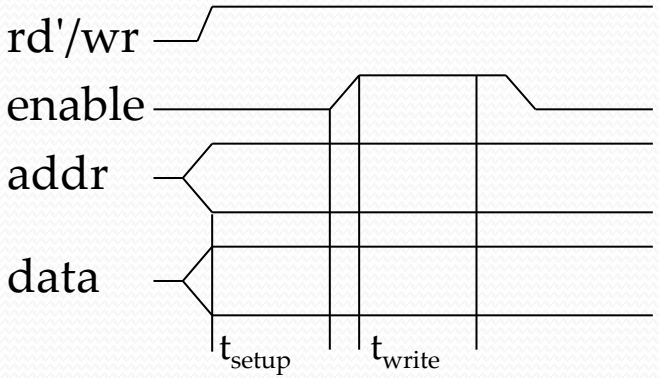
- **Diagramas de tempo** são utilizados para descrever um protocolo de comunicação.
  - Elementos básicos:
    - Tempo corresponde ao eixo horizontal.
    - Sinais de controle (ativo ALTO ou ativo BAIXO).
    - Sinais de dados (válido ou inválido).

Introdução

- **Exemplo:**



Protocolo de leitura



Protocolo de escrita

Conceitos básicos

- **Atores:**
  - Mestre: capaz de iniciar a comunicação com um dispositivo.
  - Servo / Escravo: apenas responde a uma solicitação.

- **Direção da transferência:** define qual ator transmite e qual recebe os dados. Mestres e escravos podem assumir tanto o papel de transmissores quanto de receptores.

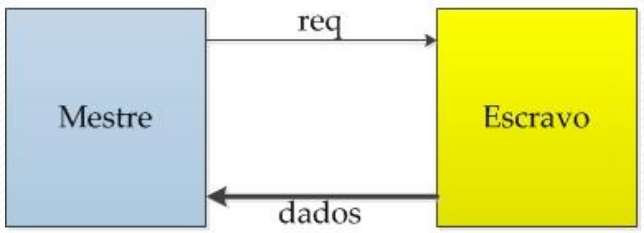
- **Endereço:** tipo especial de dado, útil para especificar a localização na memória, em um periférico ou em um registrador especial dentro de um periférico.

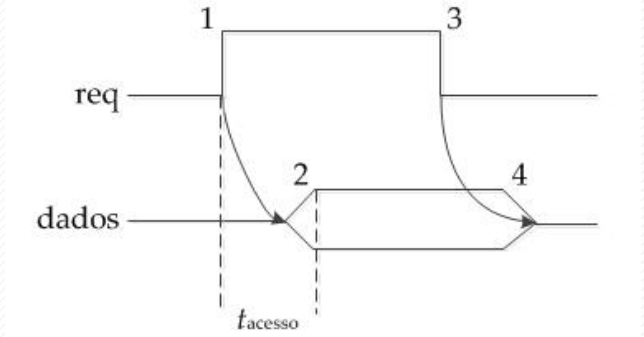
- **Multiplexação no tempo:** permite compartilhar um mesmo conjunto de conexões para a transmissão de informações diferentes.
  - Ex.: Multiplexação de endereços em DRAMs.

Conceitos básicos

- **Métodos de controle:** esquemas para organizar o início e o encerramento da transferência de dados.

A) Protocolo baseado em comandos (*strobe*): o mestre usa uma linha de controle para iniciar a transferência de dados, a qual deve ocorrer dentro de um intervalo específico de tempo (t_{acesso}). Após esse intervalo, o mestre sinaliza o fim da transmissão desativando a linha de controle.

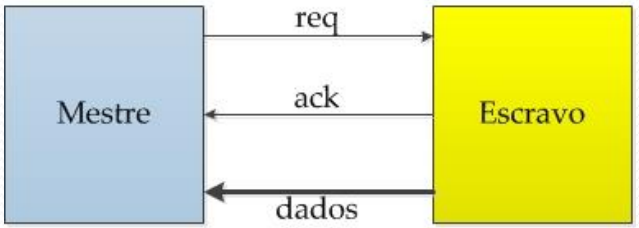


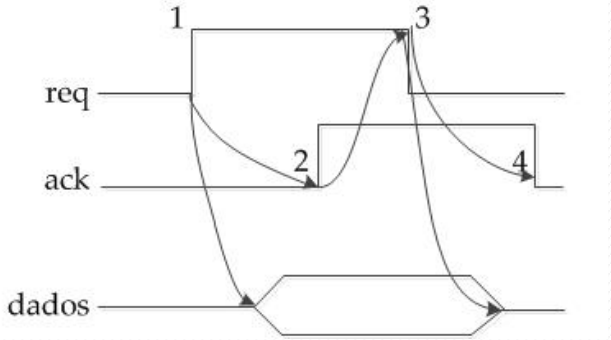


Conceitos básicos

- **Métodos de controle:** esquemas para organizar o início e o encerramento da transferência de dados.

B) Protocolo baseado em confirmação (*handshake*): o mestre usa uma linha para iniciar (e finalizar) a transferência. O escravo utiliza uma linha de confirmação para informar o dispositivo mestre o momento em que os dados estão prontos.



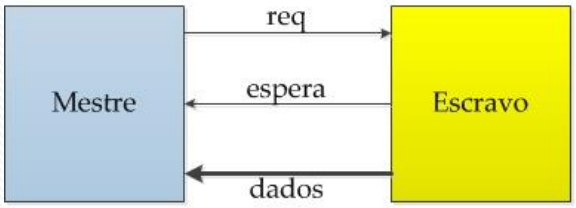


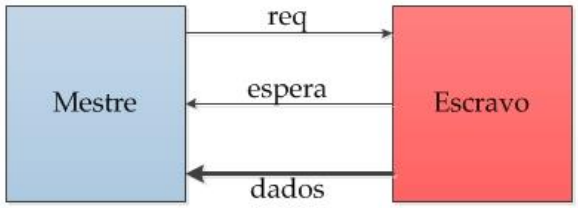
Boa estratégia para lidar com escravos que apresentem tempos de resposta diferentes.

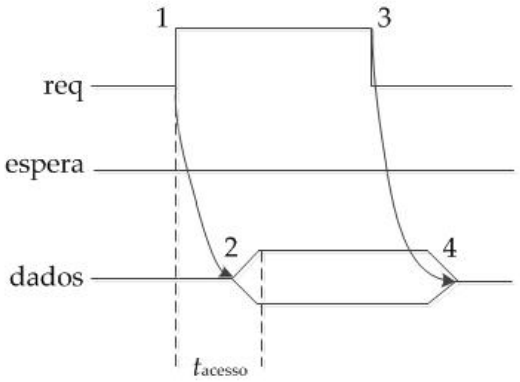
Conceitos básicos

- **Métodos de controle:** esquemas para organizar o início e o encerramento da transferência de dados.

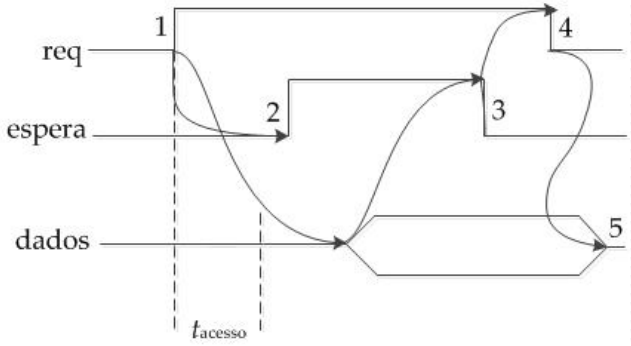
C) Protocolo misto (*strobe/handshake*):







Resposta rápida



Resposta lenta

➢ Se o escravo consegue colocar o dado dentro do tempo estipulado, o protocolo é idêntico ao *strobe*.

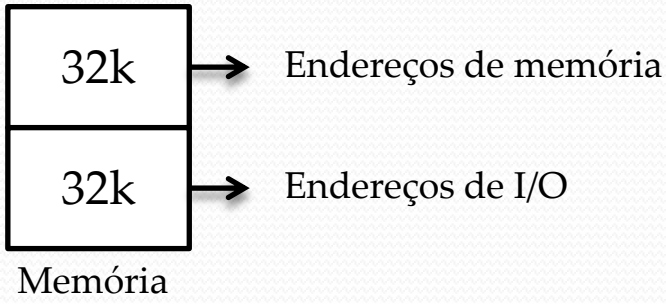
➢ Caso contrário, ele sinaliza ao mestre para que espere um pouco. Quando ele consegue disponibilizar o dado, ele retira o sinal de espera e o mestre pode fazer a leitura.

Endereçamento de I/O

- A comunicação entre um processador e outros dispositivos é feita utilizando alguns de seus pinos.
- O processador possui linhas dedicadas ao tratamento de dados, endereços e sinais de controle que formam um único barramento.
- O protocolo de comunicação é implementado em *hardware* no processador.
- Uma instrução permite a execução do protocolo de leitura / escrita no barramento.

Endereçamento de I/O

- **Tipos de endereçamento a periféricos no barramento:**
  - I/O mapeada em memória
    - Os periféricos (seus registradores) ocupam endereços específicos do espaço de endereçamento de memória existente.
    - **Exemplo:** 16 bits de endereço



  - I/O padrão (I/O mapeada em I/O)
    - Pinos adicionais no barramento indicam se o endereço fornecido deve ser associado a um acesso à memória ou a um dispositivo de I/O.
    - **Exemplo:** 16 bits de endereço

M-I/O = 0 64k de endereços correspondem à memória

M-I/O = 1 64k de endereços correspondem aos periféricos

Endereçamento de I/O

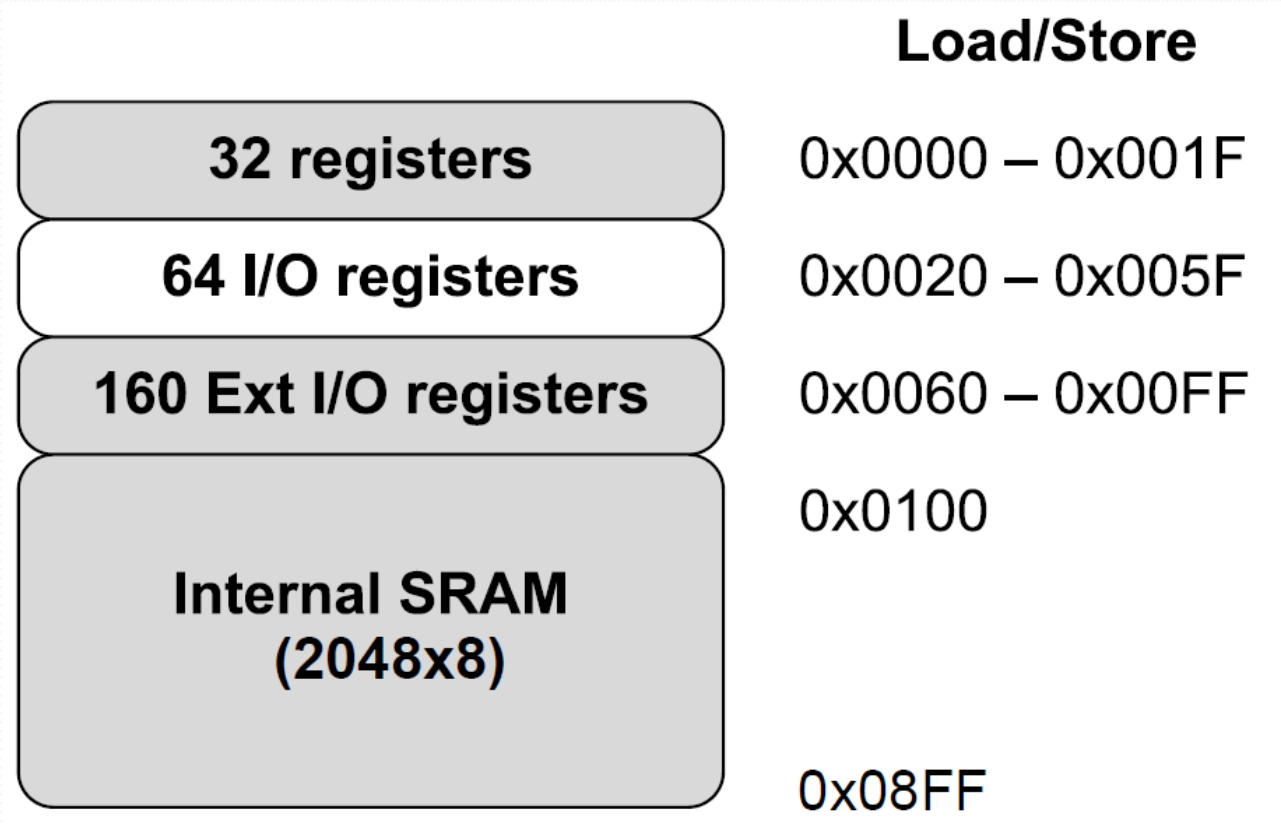
- **Comparação:**

  - I/O mapeada em memória:
    - Não necessita de instruções especiais para acessar os dispositivos periféricos – comandos como MOV e ADD também funcionam.
    - Perco espaço de memória de dados para tratar os periféricos.

  - I/O padrão:
    - Preciso de instruções especiais (e.g., IN e OUT) para mover dados entre os registradores dos periféricos e a memória / processador.
    - Não perco espaço da memória.
    - Decodificação de endereços é mais simples: quando o número de periféricos é muito menor que o espaço de endereçamento, então os bits de endereço mais significativos podem ser ignorados (decodificação parcial de endereço).

Endereçamento de I/O

- **ATmega328P:**
  - O ATmega328P usa I/O mapeada em memória:



Interrupções

- Outro aspecto importante de I/O envolve o conceito de **interrupção**.

- Considere que um microprocessador deve realizar, entre outras tarefas, a leitura e o processamento de dados de um periférico toda vez que este periférico (e.g., teclado) tiver um novo dado.

- O instante de chegada de novos dados é absolutamente imprevisível da perspectiva do microprocessador / programa em execução.

Interrupções

- **Primeira opção:** varredura (*pooling)* – durante a execução do programa, ocorrem chamadas para uma rotina que checa se o periférico tem novos dados.
  - Fácil implementação.
  - Desperdício de ciclos de execução do processador.

- **Alternativa:** o processador suporta interrupções externas – um pino especial (denominado INT) indica para o processador quando uma solicitação de atendimento foi feita por um dispositivo externo.
  - Neste caso, o processador suspende o programa em execução e desvia para a rotina de serviço de interrupção (*interrupt service routine,* ISR).

Interrupções

- Qual o endereço da ISR?
  - **Interrupção fixa:** o endereço de memória referente ao local em que a ISR deve ficar é conhecido e "programado" dentro do processador, não sendo possível modificá-lo.
    - O programador pode colocar a rotina de serviço naquele endereço ou uma instrução de desvio para a ISR.
    - É necessário haver múltiplos pinos para suportar interrupções por diferentes periféricos.
    - Ou, então, combino interrupção com varredura dentro da ISR para identificar qual foi o dispositivo que fez a solicitação.

Interrupções

- Qual o endereço da ISR?
  - **Interrupção fixa**

1. O processador executa o programa principal;
2. O periférico ativa o sinal INT para solicitar o serviço de interrupção ao processador;
3. Após completar a instrução atual, o processador percebe que INT está ativo. Ele, então, salva o estado do programa na pilha (pelo menos o PC) e carrega o PC com o valor fixo de endereço da ISR (esse valor depende do processador);
4. O processador executa a ISR;
5. Após ter sua solicitação atendida, o periférico desativa o sinal INT;
6. O processador acessa a pilha, restaura o contexto do programa principal e segue sua execução.

Interrupções

- Qual o endereço da ISR?
  - **Interrupção vetorizada:** um único pino (INT) compartilhado por vários periféricos.
    - O processador detecta a solicitação de interrupção (INT ativado) e sinaliza ao periférico, através do sinal INTA (*Interrupt Acknowledge*), para que passe o endereço da ISR pelo barramento de dados.
    - O periférico precisa ser configurado com o endereço da ISR.

Interrupções

- Qual o endereço da ISR?
  - **Interrupção fixa**

1. Processador executa o programa principal;
2. O periférico ativa o sinal INT para solicitar o serviço de interrupção ao processador;
3. Após completar a instrução atual, o processador percebe que INT está ativo. Ele, então, salva o estado do programa na pilha (pelo menos o PC) e ativa o sinal INTA (*Interrupt Acknowledge*);
4. O periférico detecta INTA ativado e coloca o endereço da ISR no barramento de **dados**;
5. O processador desvia para o endereço fornecido pelo periférico e executa a ISR;
6. Após ter sua solicitação atendida, o periférico desativa o sinal INT;
7. O processador acessa a pilha, restaura o contexto do programa principal e continua sua execução.

Interrupções

- Qual o endereço da ISR?
  - Solução de compromisso: **tabela de endereços de interrupção**.
    - Uma tabela, armazenada em memória, contém os endereços das ISRs.
    - Os periféricos não fornecem o endereço da rotina, mas o índice para acesso à tabela. Assim, poucos bits são enviados pelo periférico via barramento.
    - Além disso, é possível alterar o endereço da ISR dedicada àquele periférico sem modificar o comportamento do periférico.

Interrupções

- **Interrupções mascaráveis:**
  - O programador pode habilitar bits que instruem o processador a ignorar os sinais de solicitação de interrupção.
  - Mecanismo importante quando executa-se um código crítico para a operação do sistema.

- **Interrupções não-mascaráveis:**
  - Um pino especial é destinado a interrupções que não podem ser desabilitadas.
  - Em geral, este pino fica reservado para o tratamento de situações drásticas.
  - **Exemplo:** requisição imediata de armazenamento de dados em memória não-volátil em caso de iminente falha de alimentação de energia.

Interrupções

- **Desvio para a ISR:**

  - Alguns microprocessadores tratam este desvio de forma semelhante a uma chamada de subrotina.
    - O estado atual do sistema (PC, conteúdo dos registradores) é salvo – operação que pode consumir centenas de ciclos de relógio.

  - Outras técnicas salvam apenas o endereço de retorno (PC).
    - Neste caso, compete ao programador da ISR salvar o conteúdo dos registradores.

Acesso direto à memória

- Suponha que um periférico traga dados externos que, primeiro, precisam ser armazenados em memória e, depois, processados.

- Usando o mecanismo de interrupção:
  - Para cada novo dado, o processador desvia para a ISR, a qual, basicamente, lê o novo dado e o coloca em uma posição de memória (este tipo de armazenamento temporário de dados é chamado de *buffering*).

  - A transferência de cada um dos dados envolve o armazenamento e a restauração do estado do processador. Essas operações consomem vários ciclos de relógio.

  - O programa em execução permanecerá em espera por muito tempo durante o processo.

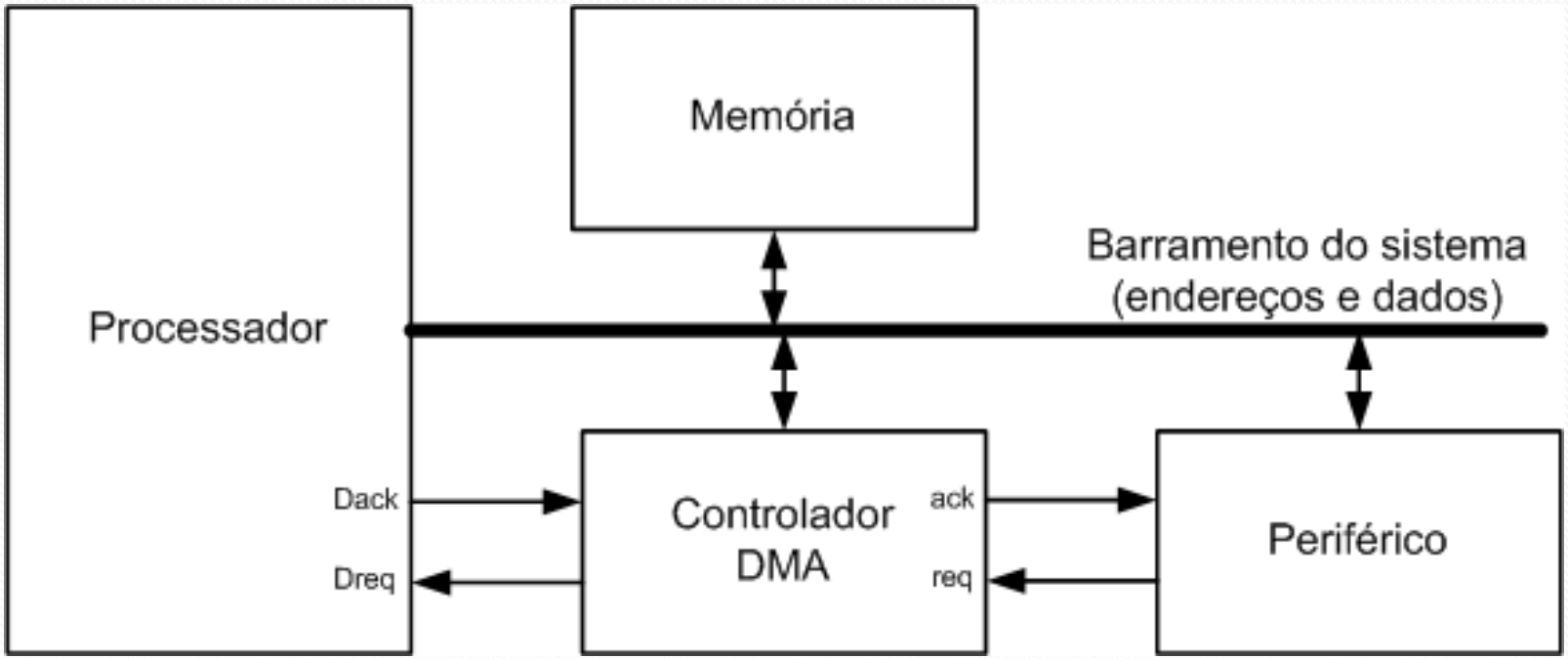
Acesso direto à memória

- **Alternativa:** permitir que o periférico se comunique diretamente com a memória, deixando o processador livre para seguir com a execução do programa.

- Um processador dedicado, chamado de **controlador DMA** (*direct memory access*), desempenha este papel.
  - Inicialmente, o processador configura o controlador DMA para que ele conheça os endereços associados aos periféricos (ou, então, estes endereços são configurados no próprio *hardware* do controlador).
  - O controlador DMA recebe uma solicitação de um periférico e, então, pede o controle do barramento ao processador.
  - O processador abdica do uso do barramento do sistema, deixando ao controlador DMA a tarefa de conduzir a transferência de dados entre o periférico e a memória.
  - Paralelamente, o processador pode seguir com a execução do programa desde que não precise usar o barramento – neste caso, o processador teria que esperar o controlador DMA encerrar sua operação.
  - Com isto, evita-se o armazenamento e a restauração do estado do programa (que ocorreriam no caso de interrupção).

Acesso direto à memória

- **Controlador DMA**



Acesso direto à memória

- **Controlador DMA**

1. Processador executa o programa principal. Os controlador DMA já foi configurado;
2. O periférico ativa *req* para solicitar atendimento ao controlador DMA;
3. O controlador DMA ativa *Dreq* requisitando o controle do barramento do sistema.
4. Após completar a instrução atual, o processador percebe que *Dreq* está ativo. Ele, então, libera o barramento do sistema colocando seus pinos em alta impedância, ativa *Dack*, e continua a execução do programa. Caso uma instrução necessite de acesso ao barramento, o processador suspende a execução e aguarda até que possa reassumir o controle do barramento;
5. O controlador DMA ativa *ack* e transfere o dado do periférico para a memória;
6. O controlador DMA desativa *Dreq* e *ack*, completando o *handshake* com o periférico.
7. O periférico desativa *req*;
8. O processador desativa *Dack* e reassume o controle do barramento do sistema.

Acesso direto à memória

- **Observações:**

  - No caso de uma arquitetura Harvard, o processador pode ler e executar instruções desde que essas não façam uso da memória de dados.

  - Um sistema com barramento exclusivo entre processador e *cache* pode continuar a execução usando apenas a *cache* enquanto o DMA faz a transferência de dados para a memória.

Arbitragem

- Qual solicitação (interrupção ou DMA) – dentre várias recebidas concomitantemente de diferentes periféricos – deve ser atendida em primeiro lugar?

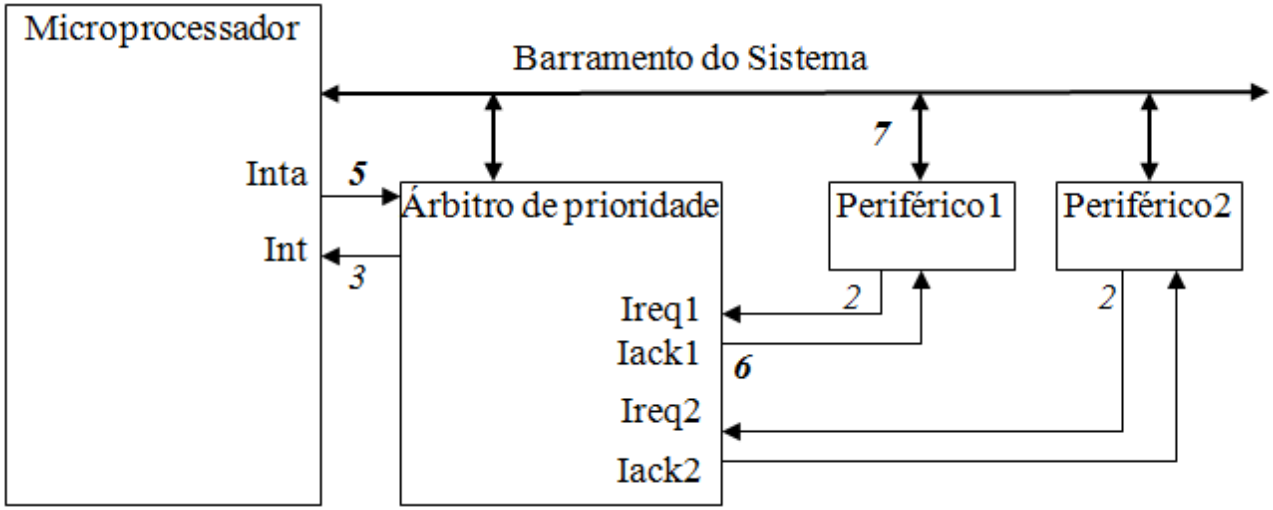
- **Arbitragem baseada em prioridades**
  - Periféricos fazem requisições para o dispositivo de arbitragem, o qual, por sua vez, faz uma solicitação ao processador.
  - De acordo com uma estratégia de definição de prioridades, o árbitro decide qual dos periféricos será atendido em sua solicitação.

    - **Prioridade fixa:** cada periférico possui uma prioridade única e pré-definida.
      - ✓ O dispositivo com maior prioridade terá a preferência no atendimento.
      - ✓ Interessante quando a diferença entre prioridades dos eventos associados aos periféricos for clara.

    - **Prioridade rotativa:** a prioridade é alterada com base no histórico de atendimento das solicitações pelo processador.
      - ✓ Melhor distribuição dos serviços entre os periféricos de prioridades semelhantes.

Arbitragem

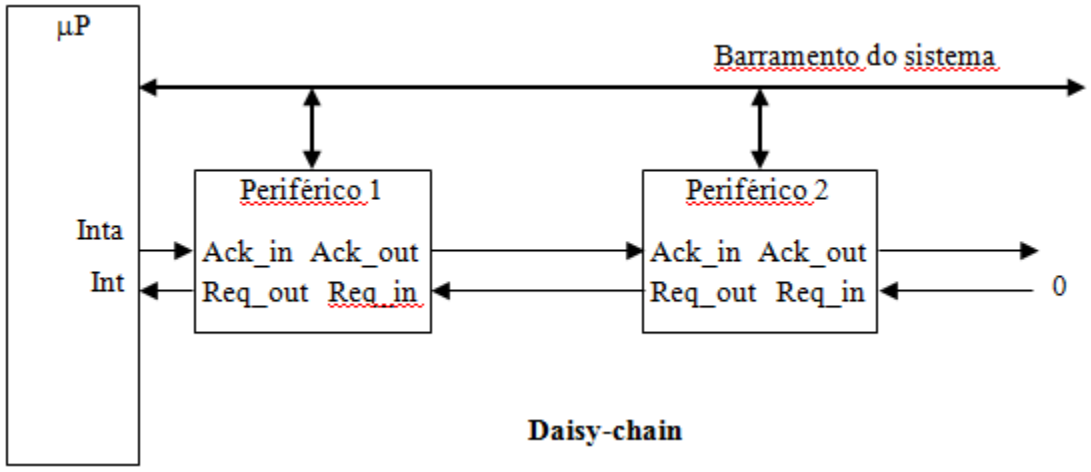
- **Arbitragem baseada em prioridades**



1. Processador executa o programa.
2. Ambos os periféricos necessitam dos serviços do processador e ativam os sinais de requisição Ireq1 e Ireq2.
3. O árbitro percebe a chegada de uma solicitação e ativa o sinal Int.
4. O processador suspende a execução do programa e armazena o estado atual.
5. O processador ativa o sinal Inta, indicando ao árbitro para prosseguir.
6. O árbitro de prioridade escolhe atender o periférico 1, ativando o sinal Iack1.
7. O periférico 1 passa a se comunicar com o processador para o atendimento da requisição via interrupção.
8. Ao final da execução da ISR associada ao periférico 1, o processador retoma a execução do programa.

Arbitragem

- **Arbitragem daisy-chain**
  - Os periféricos são conectados em uma lista e repassam sinais de requisição e confirmação nas respectivas direções.
    - Sinais de requisição são repassados entre os periféricos em direção ao processador.
    - Sinais de confirmação também são repassados, mas do processador em direção aos periféricos.
    - Os periféricos mais próximos do processador possuem maior prioridade.



Daisy-chain

Arbitragem

- **Características:**
  - Facilidade de acrescentar ou remover periféricos sem ter que reprojetar o sistema.
  - Não suporta esquemas avançados de arbitragem, como a estratégia de prioridade rotativa.
  - A falha de um periférico na cadeia pode causar a perda de acesso a outros periféricos.
  - A presença de um número elevado de periféricos na cadeia pode tornar o tempo de resposta à requisição muito lento.

Múltiplos barramentos

- Um único barramento para todas as comunicações entre os dispositivos pode não ser uma boa opção:
  - Periféricos necessitariam de um processador específico de alta velocidade para a interface com o barramento – excesso de portas lógicas, maior consumo de potência e custo.
  - A presença de muitos periféricos pode sobrecarregar o barramento deixando-o mais lento.

- **Ideia:** utilizar dois níveis de barramento.
  - Barramento local do processador: alta velocidade e número maior de bits para a comunicação com *cache*, memória, processadores, etc.
  - Barramento de periféricos: mais lento, com um número menor de bits e menos frequentemente acessado.
    - Normalmente segue padrões para portabilidade (ISA, PCI, etc.)
  - Ponte (*bridge*): processador dedicado que converte a comunicação entre os barramentos.

Múltiplos barramentos

- **Exemplo**