Organização e Arquitetura de Processadores

Modos de Transferência

Introdução aos Modos Bloqueado, *Polling* e Interjeição

Capítulo 10.5 do Monteiro
Capítulo 7 do Stallings

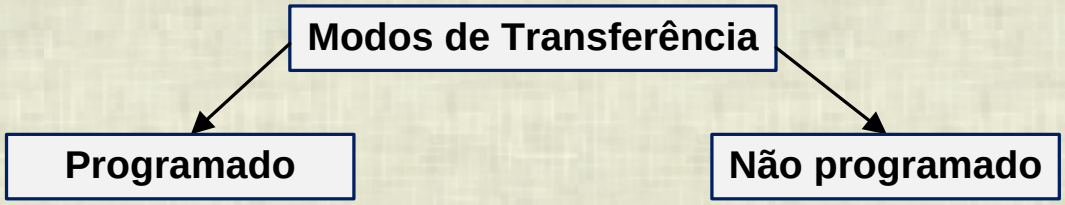
Última alteração: 15/06/2022

Prof. Ney Laert Vilar Calazans

Baseado em notas de aulas originais do Prof. César Augusto Missio Marcon

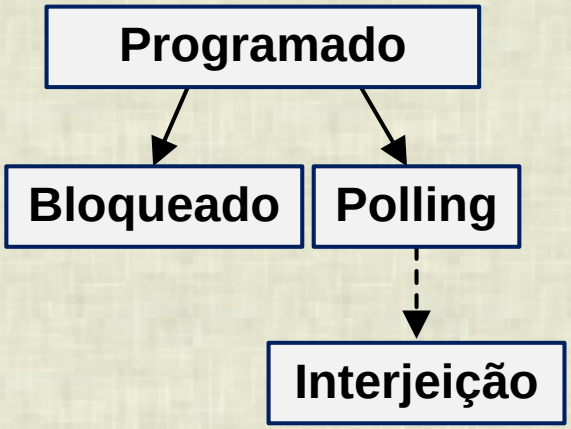
Introdução

- **Motivadores → analisar modos de transferência**
  - Quando efetuar a transferência (atendendo requisitos de projeto)?
  - Mecanismos para controlar transferências (eficientemente)?
- **Requisitos**
  - Resposta rápida a eventos críticos
    - Eventos de segurança exigem resposta imediata
    - Eventos em tempo real → têm tempo máximo para receber resposta (*deadline*)
  - Não sobrecarregar CPU com E/S
    - Acesso à disco e *refresh* de memória podem exigir muito serviço
  - Modos básicos (são dois)



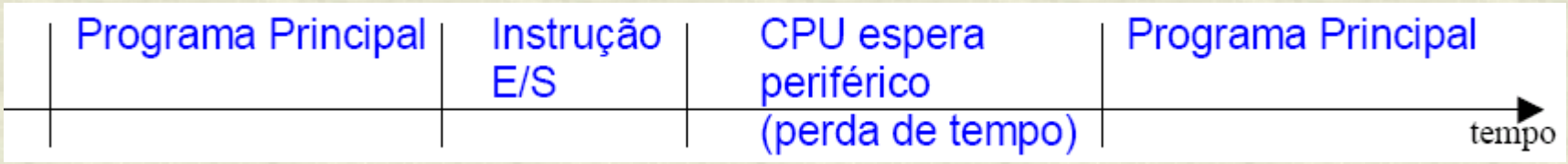
Entrada/Saída Programada

- **E/S controlada pela CPU**
- **Procedimento**
  1. Em instante determinável, processador pergunta a periférico se este está apto a receber ou transmitir
  2. Em caso afirmativo, realiza transferência
- **Tipos**



Modo Bloqueado

- **Uma vez iniciada a comunicação, CPU fica bloqueada**
  - Ocupada e escrava do periférico até terminar a operação
- **Problema**
  - Periféricos, em geral → muito mais lentos que CPU
- **Consequência**
  - CPU é sub-utilizada

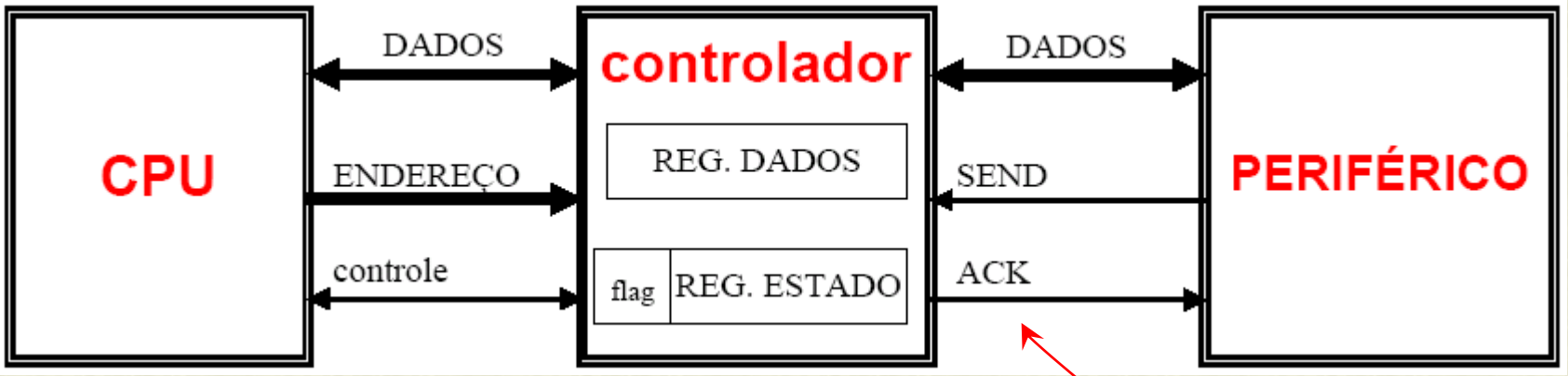


- **Exemplo de utilização**
  - Máquinas reativas → Esperam ações externas captadas por sensores para então responder ao meio
  - Computador dedicado a tarefas dependentes de um único periférico
    - E. g. pesagem → Única atividade da máquina é pesar

Polling

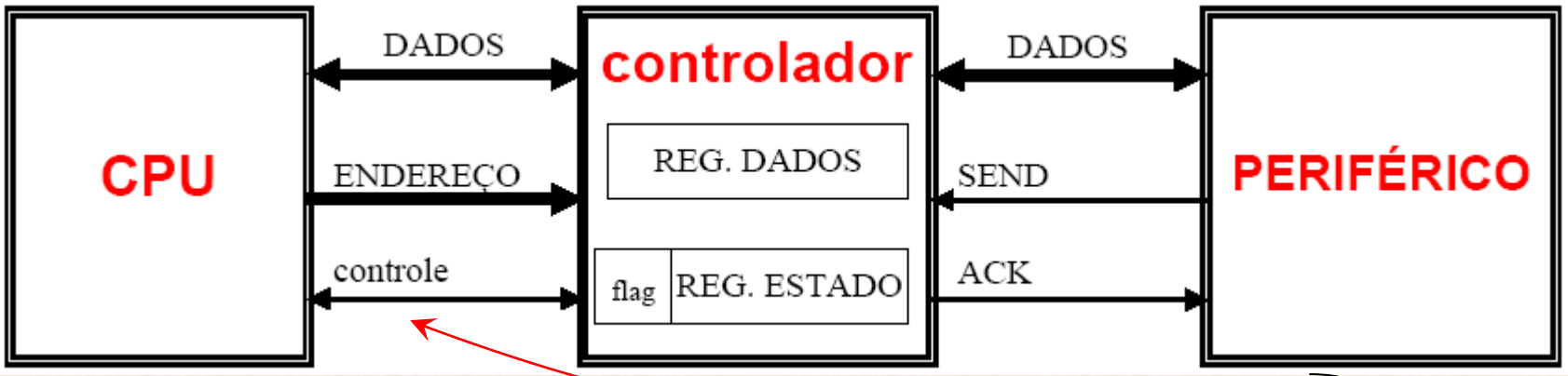
- **CPU possui controle total da comunicação**
  - Determina instantes de tempo qdo ocorrem transferências
- **CPU periodicamente testa se há dispositivo p/ se comunicar**
  - A pergunta pode ser diretamente em um porta de entrada da CPU
- **Otimização pode ser feita com controladores**
  - CPU periodicamente testa registradores de estado dos controladores de E/S
    - Registradores sinalizam possíveis transferências a realizar

Polling - Controlador e Periférico



1. Periférico transfere dados para controlador
   - Coloca dados no barramento
   - Ativa sinal *send*
2. Controlador armazena dados
   - Dados lidos são colocados em *buffers*
   - Responde *ack* para periférico
   - Sinaliza “dado presente” para CPU, ativando *flag*
3. Controlador habilita novo envio de dados
   - Remove *ack* se
     - *Flag* desativada (qdo CPU já buscou) ou
     - Tem espaço no *buffer* local para mais dados
4. Periférico habilita novo envio de dados
   - Remove *send* ao detectar remoção de *ack*

Polling – CPU e Controlador



1. CPU verifica se tem dados a transferir
   - Executa *polling* periodicamente no registrador de estado
2. CPU lê o dado do controlador
   - Quando *flag* ativo
     - Lê dado do barramento
     - Armazena dado em memória
3. CPU avisa controlador de leitura com sucesso
   - CPU envia sinal de controle para desativar *flag*
4. Periférico pode enviar novo dado

Polling – Overrun Error

- **No protocolo descrito no *slide* anterior dados só são enviados quando *ack* desativado. Assim, o que acontece quando um periférico necessita enviar uma rajada de dados?**
  - Tem que esperar
- **Otimização no protocolo**
  - Permitir que periférico envie novos dados assim que receber *ack*
- **Condição**
  - Se CPU demorar muito para ler os dados recebidos (mais tempo do que a nova escrita de dados)
- **Motivos possíveis**
  - CPU mais lenta que o periférico
  - CPU ocupada com outras tarefas mais prioritárias
- **Problema**
  - Dados novos podem sobrescrever dados que ainda não foram lidos
- **Como amenizar problema?**
  - Área de armazenamento temporário (*buffer*) no controlador

Polling – Exercício de *Overrun Error*

1. **Dada uma serial assíncrona de 2 Mbps sendo escrita em um buffer de um controlador, qual a área de armazenamento deste *buffer* para que um programa possa ler este por *polling*, sem perder dados, a taxa máxima de leitura uma a cada 100ms?**
2. **Faça um esquema temporal que ilustre os tempos de escrita e leitura de dados no *buffer***
3. **Faça um esquema de blocos ilustrando os principais sinais na comunicação**

Polling – Cálculo de Tempo Desperdiçado

- **Nos slides seguintes, analise o tempo desperdiçado para atender por *polling* os requisitos especificados em cada caso**

Polling – Cálculo de Tempo Desperdiçado 1

- **Dados**
  - CPU lê *flag* a cada 0,1 µs (10 MHz)
  - E/S transfere dados a 1 KB/seg
  - Largura do barramento: 1 Byte

B= byte (8 bits)

- **Pergunta**
  - Quantas leituras desnecessárias são feitas ao *flag* para realizar uma transferência?
- **Resposta**
  - E/S transfere 1 B/ms

$$\frac{1ms}{0{,}1\mu s} = \frac{10^{-3}}{10^{-7}} = 10^4$$

- ***9999 leituras sem sucesso para 1 com sucesso***

Polling – Cálculo de Tempo Desperdiçado 2

- **Dados**
  - CPU operando a 50 MHz
  - Rotina de *polling* consume 100 ciclos
- **Pede-se**
  - Tempo de CPU consumido para verificar um *mouse*, sabendo-se que este deve ser verificado 30 vezes/s
- **Resposta**
  - Quantidade de ciclos gastos pela CPU para realizar todas as transferências
    - 30 × 100 = 3.000 ciclos/s
  - % de ocupação da CPU
    - (3.000 × 100%) / 50×10^6 = 0,006%
- **Conclusão**
  - *mouse* não incomoda

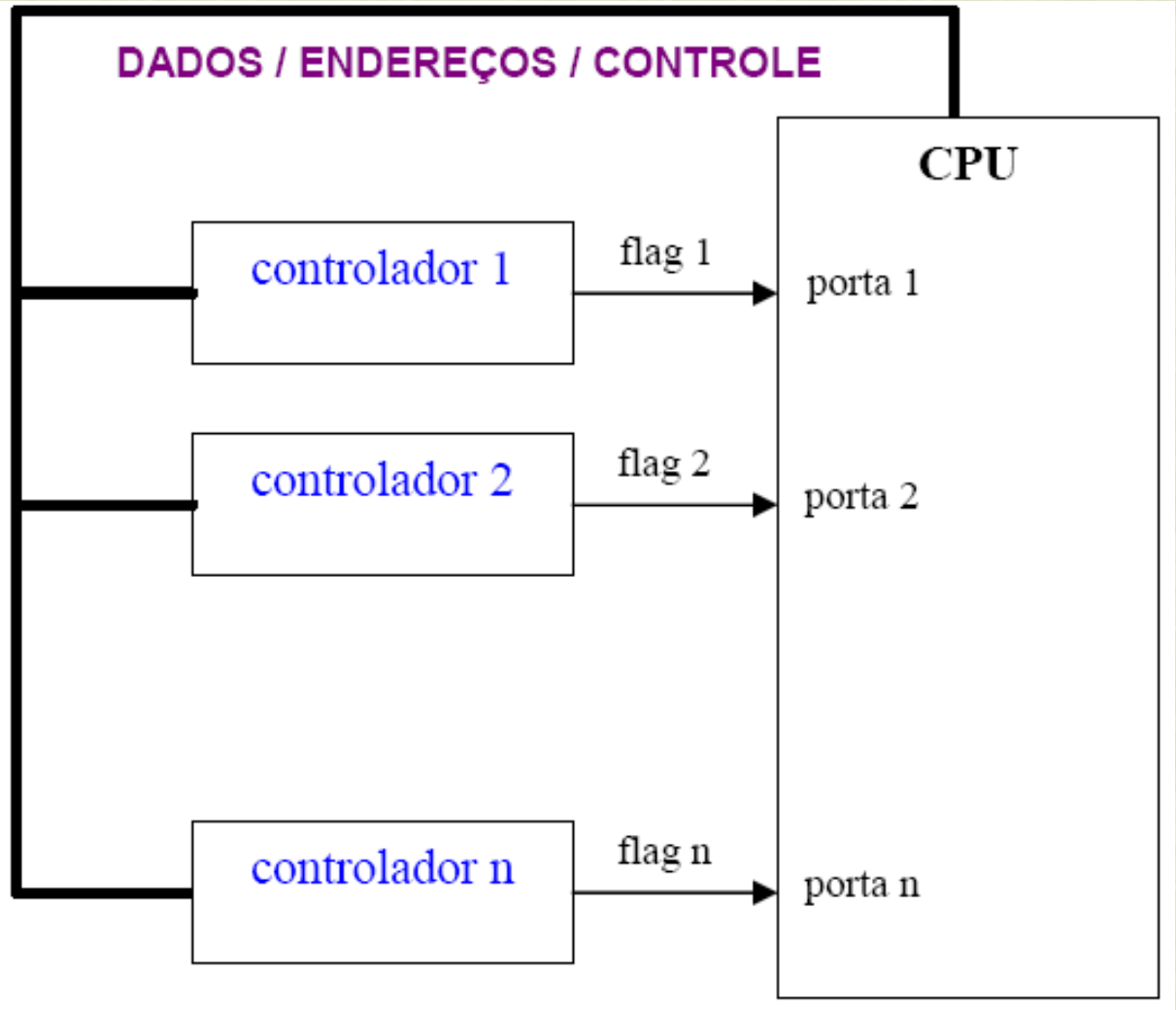
Polling – Cálculo de Tempo Desperdiçado 3

- **Condições**
  - CPU operando a 50 MHz
  - Rotina de *polling* consume 100 ciclos
- **Pede-se**
  - Tempo para transferir dados de um periférico
  - Taxa de 50 KB/s, transferindo-se 2 bytes por vez
- **Resposta**
  - Quantidade de transferências por segundo
    - $(50 \times 10^3$ Bytes / 2 Bytes) = 25.000 transferências/s
  - Quantidade de ciclos gastos pela CPU em todas transferências
    - 25.000 × 100 = 2.500.000 ciclos/s
  - % de ocupação da CPU
    - $(2.500.000 \times 100\%) / 50 \times 10^6 = 5\%$
- **Conclusão**
  - Transferência de dados é tolerável

Polling – Cálculo de Tempo Desperdiçado 4

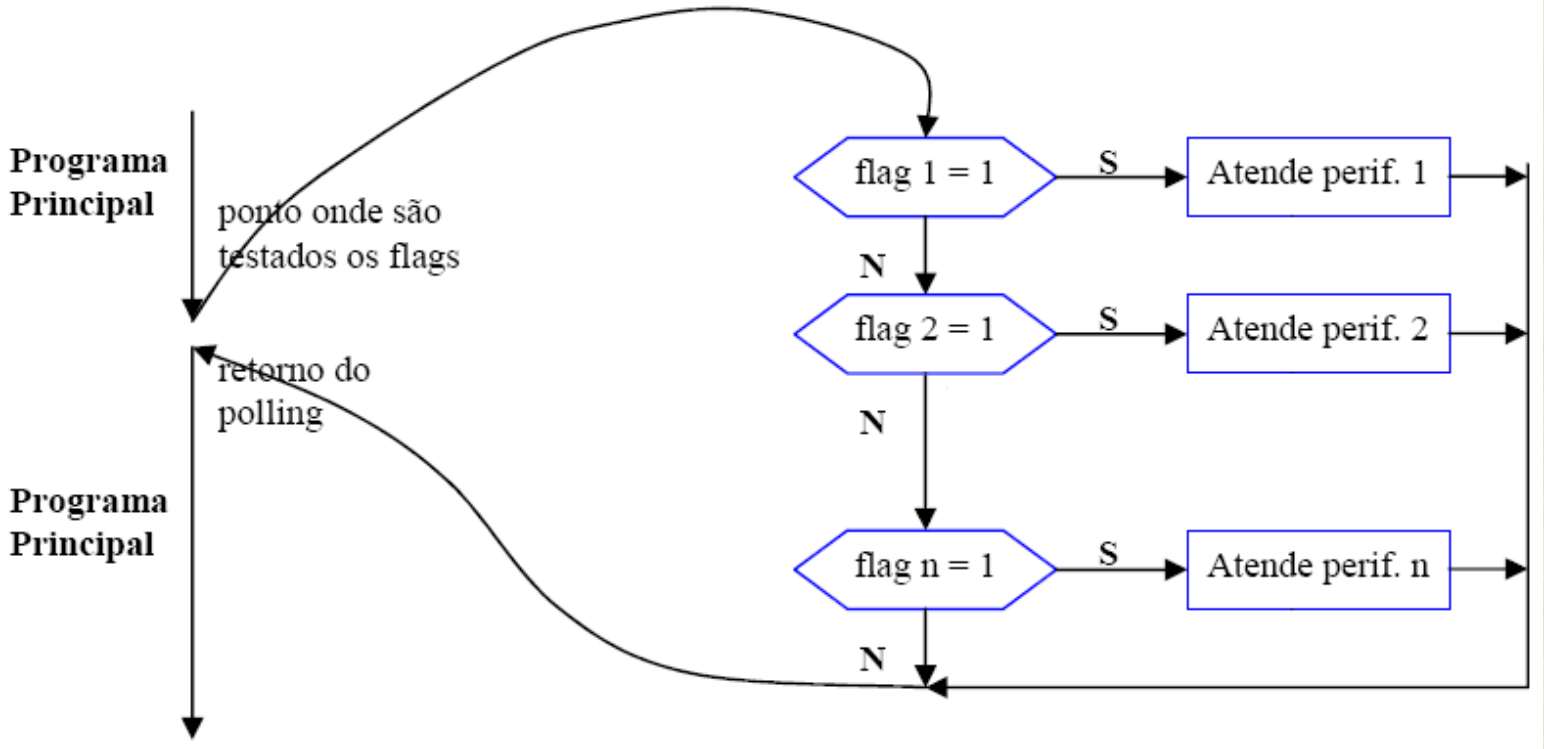
- **Condições**
  - CPU operando a 50 MHz
  - Rotina de *polling* consume 100 ciclos
- **Pede-se**
  - Tempo para transferir dados de um disco rígido
  - Taxa de 2 MB/s, transferindo 4 bytes por vez
- **Resposta**
  - Quantidade de transferências por segundo
    - $(2\times10^6$ Bytes / 4 Bytes) = 500.000 transferências/s
  - Quantidade de ciclos gastos pela CPU em todas transferências
    - $500.000 \times 100 = 50\times10^6$ ciclos/s
  - % de ocupação da CPU
    - $(50\times10^6 \times 100\%) / 50\times10^6 = 100\%$
- **Conclusão**
  - A CPU deveria ficar totalmente dedicada a tratar transferência de dados do disco rígido através de *polling*

Polling em Vários Dispositivos 1



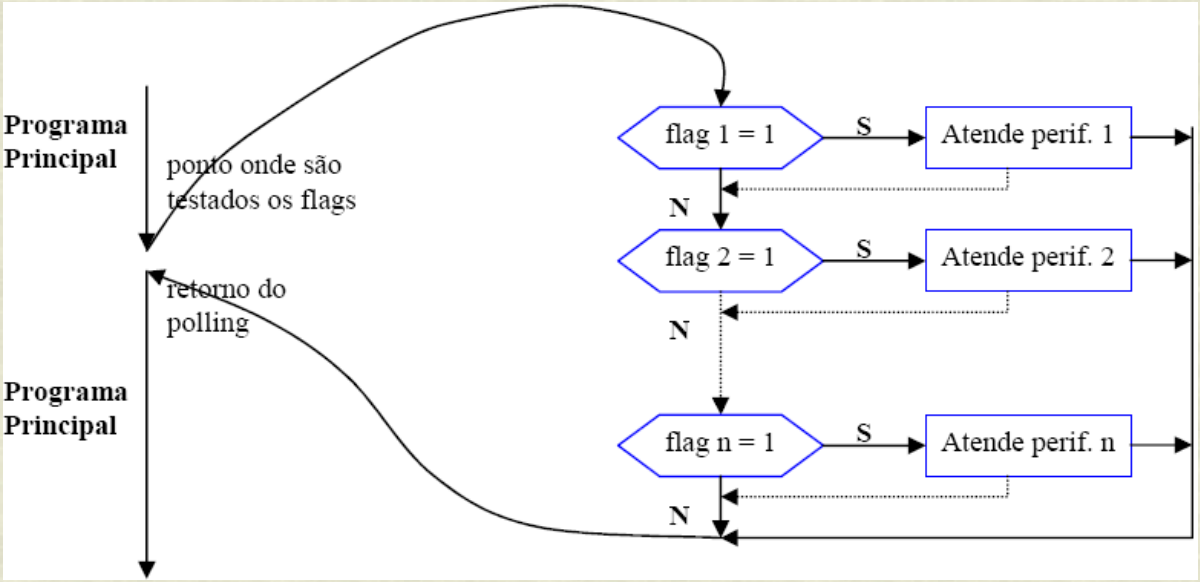
Polling em Vários Dispositivos 2

- **Verificação de *flags***
  - Prioridade implícita
  - Normalmente atende apenas um periférico por *polling*
  - Prioridade de atendimento ao menor número de periférico pode acontecer *starvation*



Polling em Vários Dispositivos 3

- **Solução para evitar *starvation***
  - Leitura de mais de um periférico



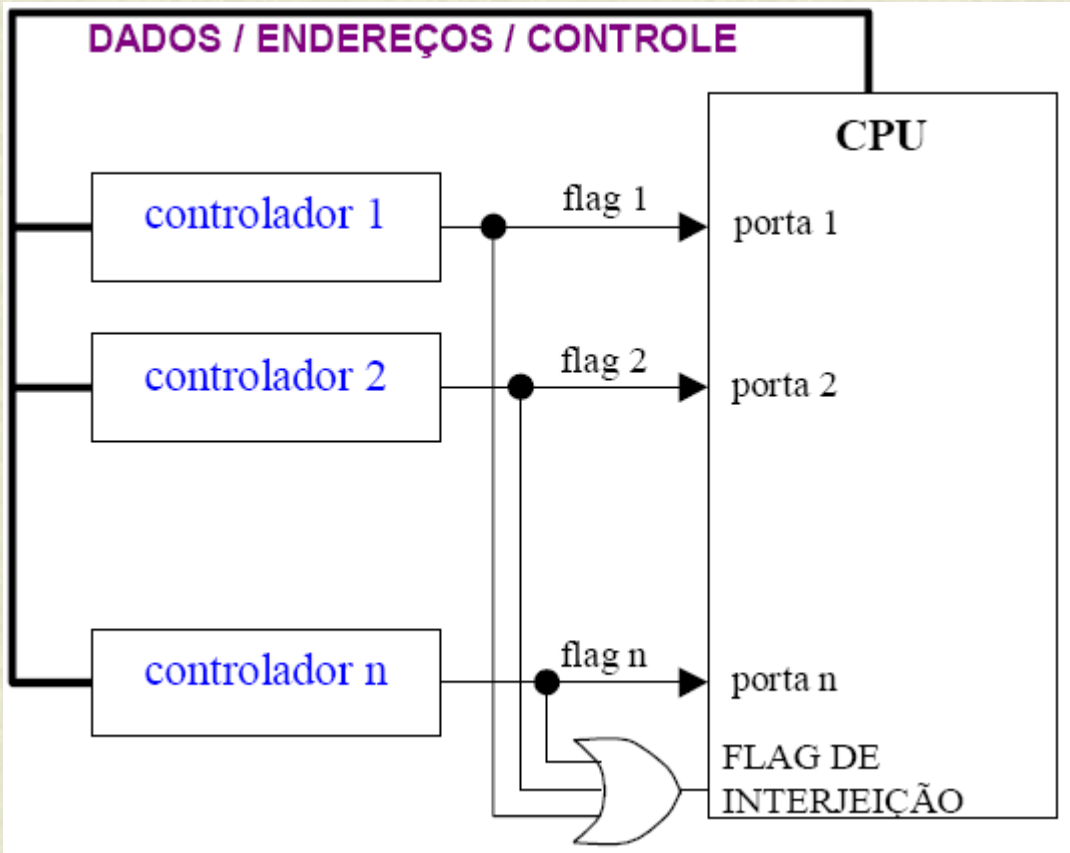
  - Esta solução pode causar muito atraso para a CPU
- **Projetista decide em função de**
  - Número de periféricos
  - Frequência estimada dos pedidos de atendimento
  - Frequência de teste dentro do programa principal
- **Existe alguma outra solução melhor?**

Polling em Vários Dispositivos 3

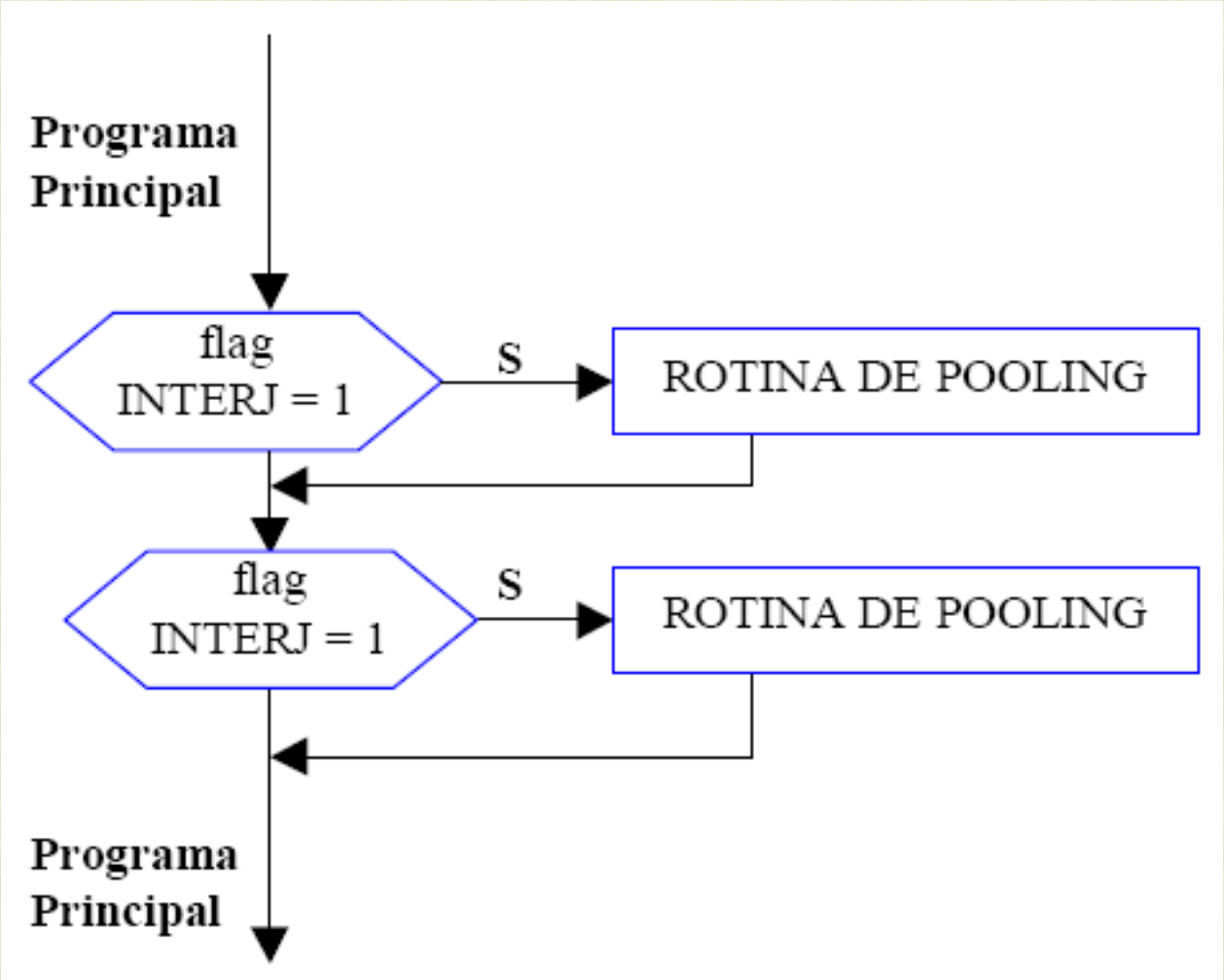
- **Existe alguma outra solução melhor?**
  1. Atendimento com fila. Quem foi atendido vai para o final da fila e tem a última prioridade no próximo *polling*
     - Analise vantagens e desvantagens
  2. ...
  3. Interjeição

Interjeição

- **O que é?**
  - Otimização do *polling*
- **Funcionalidade**
  - Ou lógico de todos os *flags*
- **Consequência**
  - Menor tempo com o teste
- **Procedimento**
  - CPU só testa um bit → independentemente do número de periféricos



Interjeição – Implementação da Funcionalidade



Exercício

1. **Faça um trecho de programa que permita ter uma comparação de software entre modo bloqueado, *polling* e interjeição**
   - Considere o programa descrito em C, com as funções
     - getchar() (função bloqueante que lê um inteiro do teclado), e
     - kbhit() (função não bloqueante que verifica se existem caracteres no *buffer* de teclado)
   - Para a comparação com interjeição, considere a possibilidade de acesso a 3 diferentes teclados
2. **Qual o método utilizado para otimizar E/S tipo *polling*? Explique-o**
3. **Suponha um programa de alto nível que avalia (por *polling*) a cada 20ms um *buffer* serial com capacidade para armazenar 64 bytes**
   - Supor que o tempo de leitura do *buffer* é desprezível
   - Qual a maior taxa em bytes/s para comunicação serial sem que ocorra *overrun*?

   RESPOSTA:
   - TaxaLeitura >= 3.200 bytes/s
   - TaxaEscrita <= 3.200 bytes/s

Exercícios

4. **Dada uma serial assíncrona de 2 Mbps sendo escrita em um buffer de um controlador**
   - Qual a área de armazenamento deste *buffer* para que um programa possa ler este por *polling*, sem perder dados, a uma taxa máxima de uma leitura a cada 100 ms?
   - Faça um esquema temporal que ilustre os tempos de escrita no *buffer* e leitura de dados
   - Faça um esquema de blocos ilustrando os sinais principais na comunicação
5. **(Com resposta) Seja dado o projeto de um sistema embarcado cujo processador acessa um dispositivo de entrada/saída por *polling* a uma taxa média de 1 acesso a cada 20 ms. Considerando uma comunicação assíncrona P82 (paridade par, 8 bits de dados e dois *stop* bits além do *start* bit), com 144.000 bps e uma UART com um registrador de buffer e um de deslocamento, sendo que em cada *polling* a CPU pode fazer uma ou duas escritas na UART, calcule:**
   - Qual a taxa máxima de transmissão ideal (não deve ser considerado o tempo de *polling* do processador)?
   - Qual a taxa efetiva de transmissão em caracteres/s?
   - Qual o tempo necessário para transmitir 100 Kbytes para cada caso acima?
   - Faça um esboço para ilustrar os instantes de tempo onde ocorre transmissão de dado, a comunicação CPU com dispositivo de E/S, e o tempo em que o processador está em atividades entre *pollings*

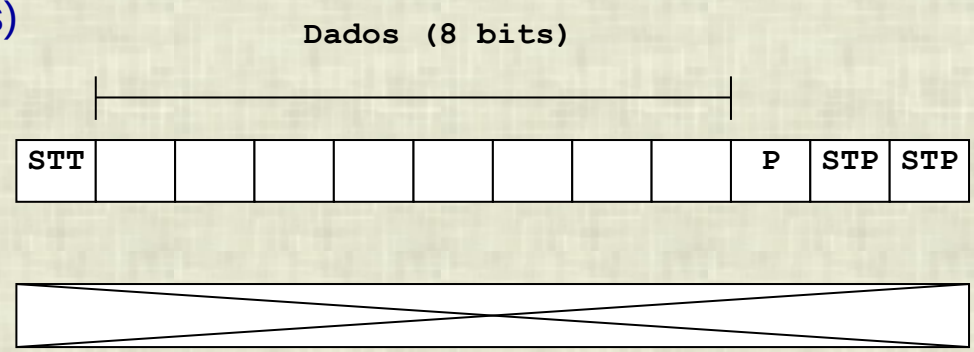
Exercícios

RESPOSTA (5):

Acesso (*polling*) = 1/20 ms Dados (8 bits)

Comunicação = P82 →

- STT – Start Bit
- P - Paridade
- STP – Stop Bit



Taxa = 144.000 bps (bits por segundo)

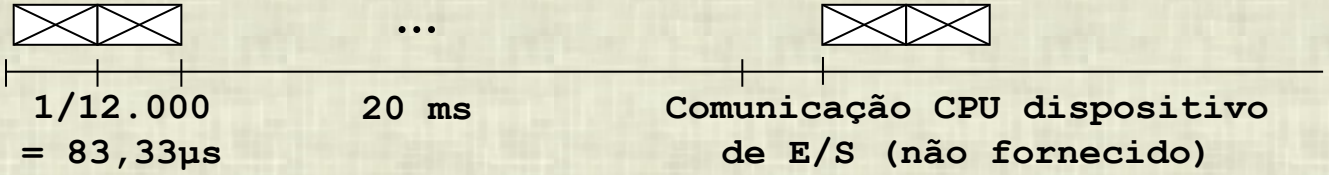
UART = 1 buffer + 1 shift register

Quantidade = 100 Kbytes (KB)

A) Taxa máxima = 144.000 bps / 12 b = 12.000 Bps (bytes por segundo)

Tempo para transmitir na taxa máxima = 100KB / 12.000 Bps = 8,333 seg

B) Taxa efetiva



C) Taxa efetiva = 2 B / 20 ms = 1 B / 10 ms = 100 B/s (desconsiderando o tempo de comunicação da CPU com dispositivos de E/S)

Tempo = 100KB / 100 Bps = 1Ks = 1000 s