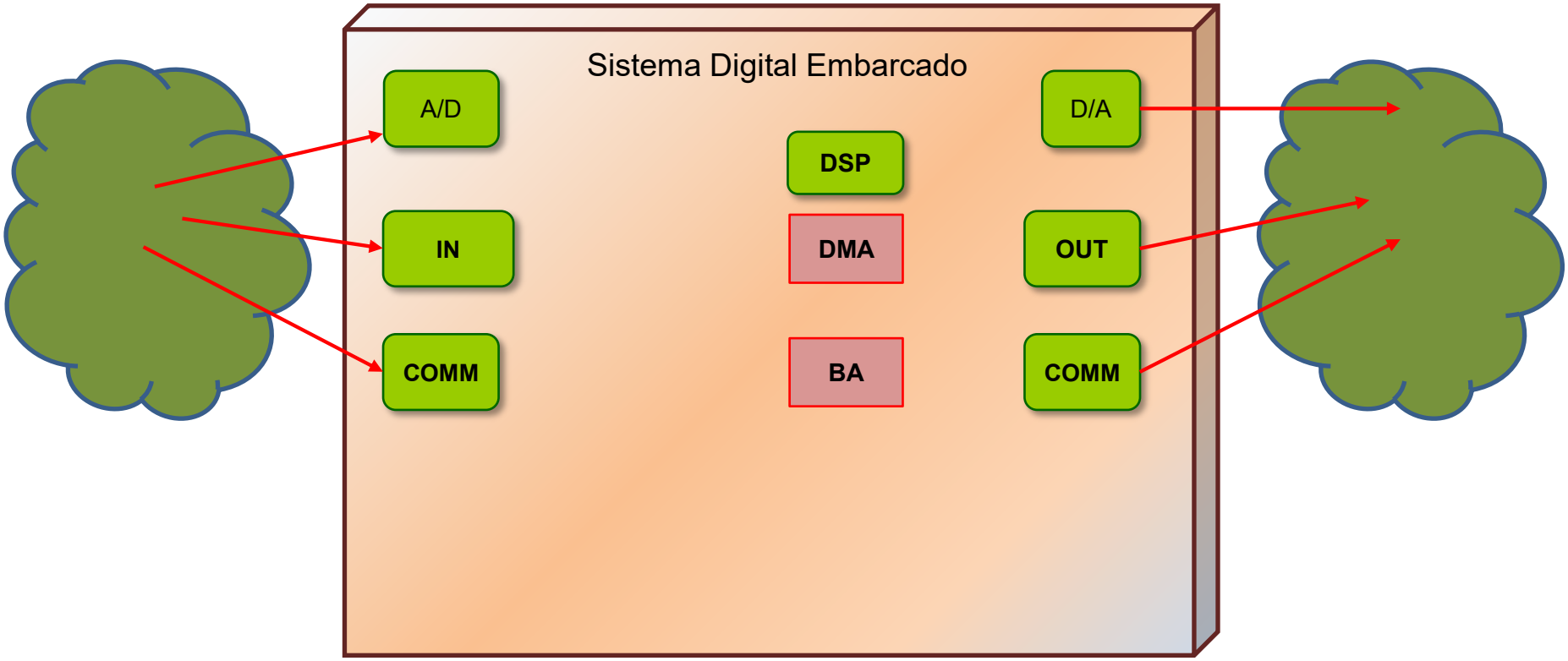
- Núcleo do ARM Cortex – hardware...

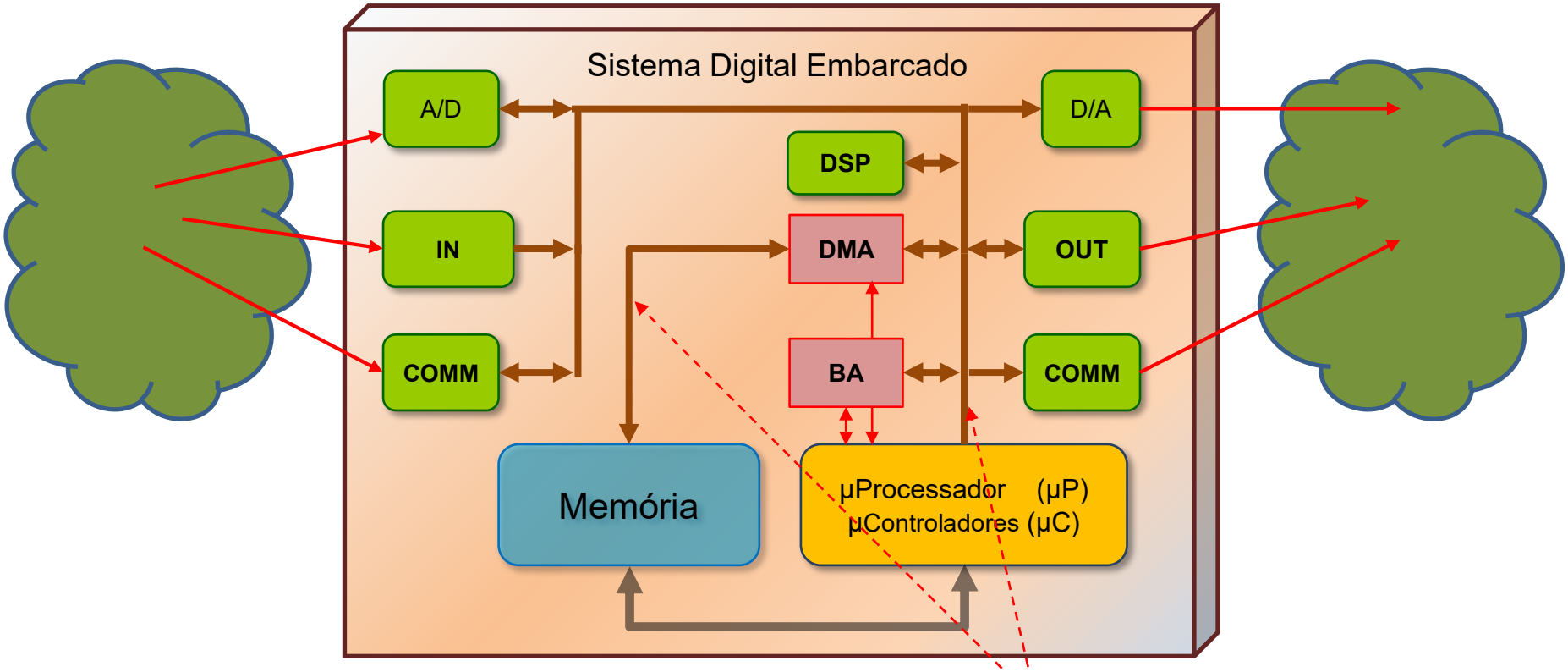
(Registradores, ULA, GPIOs, pipeline, barramentos e interconexão com vários periféricos)

- Conceitos de programação – e.g. superloop, funções, assembly...
- Conceito de Interrupção – programação do NVIC
- Programação de periféricos (GPIO, timers, Systick, PWM...)

- HOJE: comunicação entre sistemas processados (comunicação entre eletrônica embarcada)
- Comunicação paralela
- Comunicação serial...



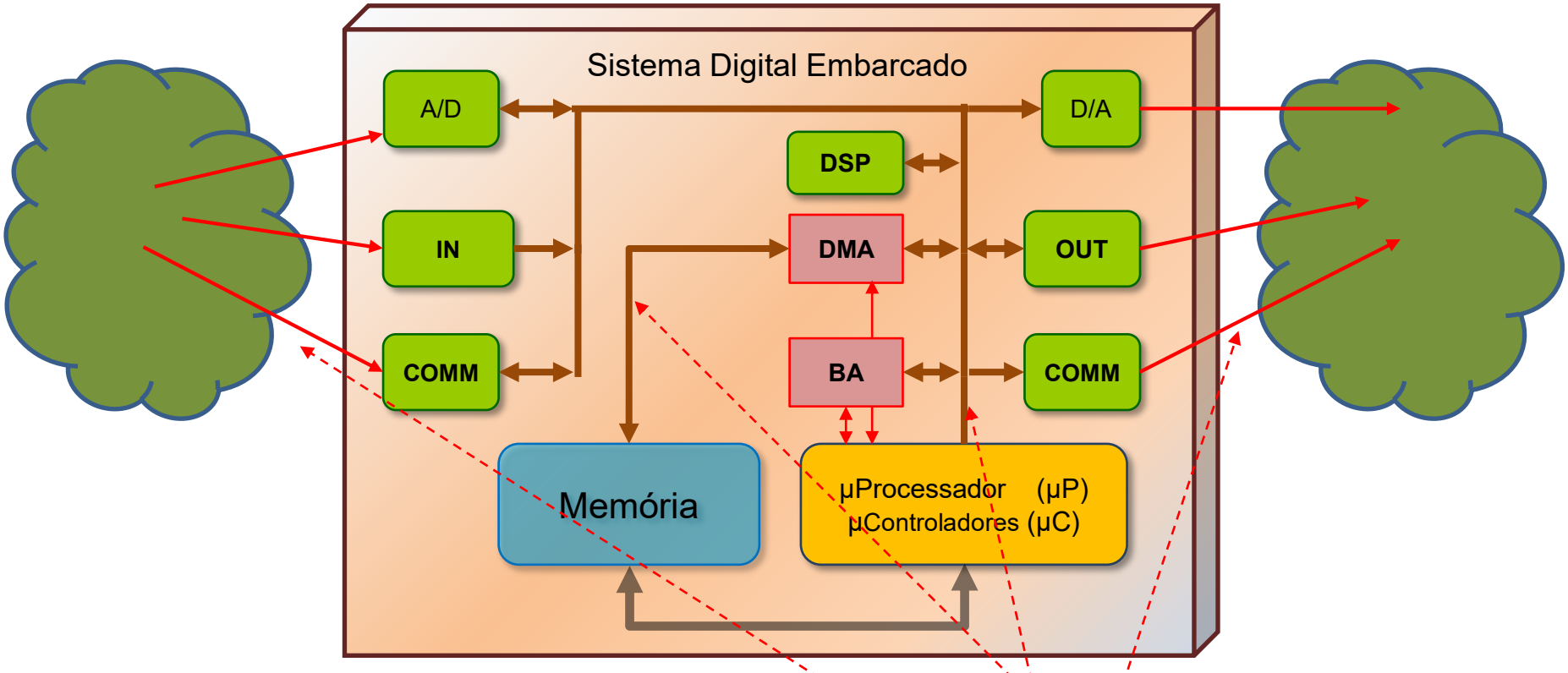
Sist. Embarcados são compostos por vários tipos de circuitos dedicados (periféricos), que precisam se comunicar com o mundo exterior e com o processador principal.



Interno ao circuito integrado, processadores, memória e periféricos usam ***barramentos*** que interconectam processadores, periféricos e memória, formando um subconjunto interno de ***interfaceamento***.

Processadores: - dedicados (standard – periféricos, ou customizados – criados pelo projetista)
- especializados (processadores de propósito específico)
- de uso geral (μProcessadores ou μControladores).

Armazenagem: implementado por sistemas de ***memória***.



Sist. embarcados são compostos por processadores, memória e periféricos interconectados internamente por ***barramentos***, associados a periféricos que permitem criar ***redes de comunicação*** com outros sistemas.

Processadores: - dedicados (standard – periféricos, ou customizados – criados pelo projetista)
- especializados (processadores de propósito específico)
- de uso geral (μProcessadores ou μControladores).

Armazenagem: implementado por sistemas de ***memória***.

Iterfaceamento: implementado por ***barramentos*** *(**paralelos**)* e/ou ***redes*** (***seriais***) entre sistemas.

Aspectos funcionais de um Sistema Embarcado:

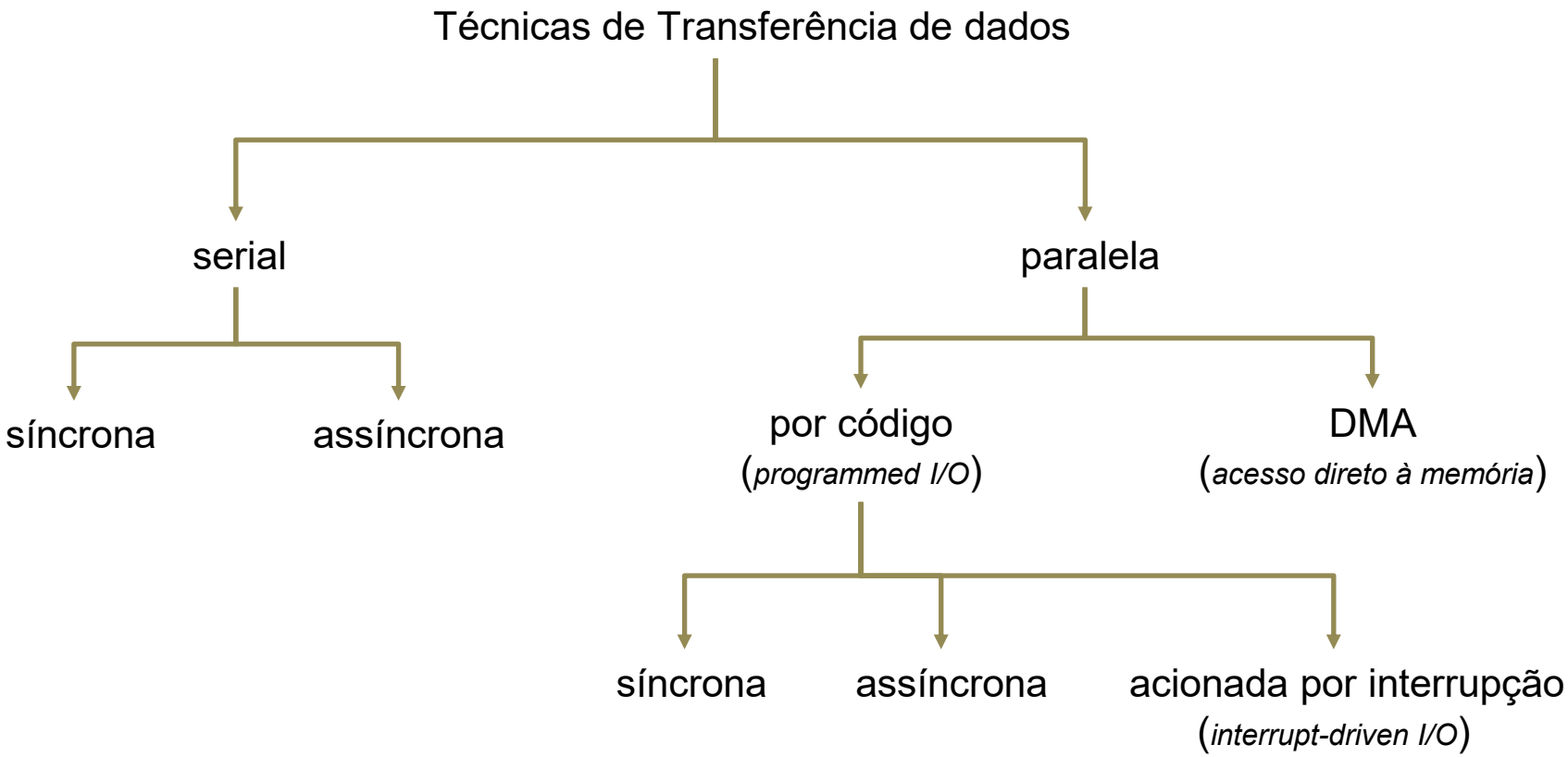
- Sensoriamento e coleta de dados (*e.g. inputs de GPIO, ADC, etc...*)
  - ✓ Transdução de algumas formas de energia em ***dados*** e ***informações*** sobre o mundo, implementado usando sensores e conversores;

- Atuadores e distribuidores de dados (*e.g. output de GPIO, PWM, LCD, etc...*)
  - ✓ Conversão de dados e informações em outras formas de energia ou transferem informações para outros sistemas, implementado usando conversores e redes;

- Armazenagem da informação (memória) (*e.g. Flash, HD, SSD, etc...*)
  - ✓ Retenção de dados e informações, implementado com sistemas de memória;

- Processamento (*e.g. DSP, ULA, circuito em FPGA, etc...*)
  - ✓ Transformações em dados, implementado com processadores;

- Comunicação (*e.g. USB, Wireless, Ethernet, I2C, CAN, etc...*)
  - ✓ Transferência de ***dados*** entre processadores e memória,
  - ✓ Transferência de dados e informações entre ***dispositivos*** embarcados,
  - ✓ Implementados por barramentos e por redes de comunicação, chamado genericamente de sistemas de *interfaceamento*;

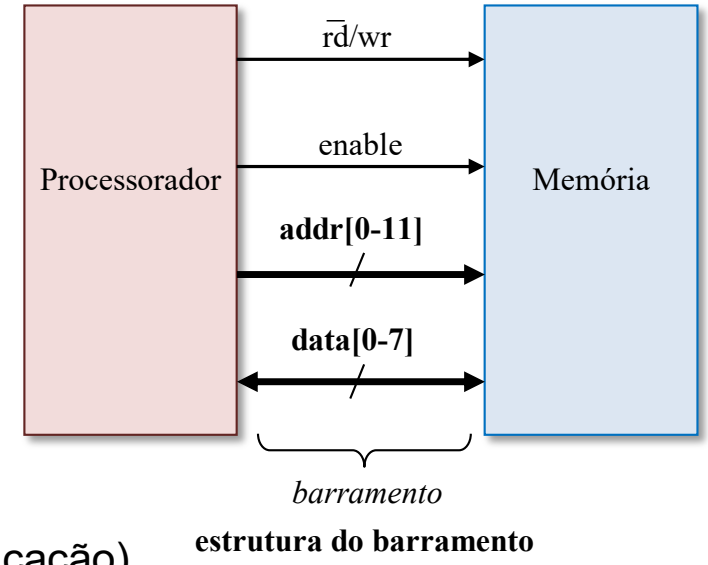

Técnicas de Transferência de dados
serial
paralela
síncrona
assíncrona
por código
(programmed I/O)
DMA
(acesso direto à memória)
síncrona
assíncrona
acionada por interrupção
(interrupt-driven I/O)

➢ Usamos a palavra ***bus*** (***barramento***) para definir duas coisas diferentes...

- ***Fio*** (*wire*):
  - Unidirecional ou bidirecional
  - Uma linha pode representar múltiplos fios

- ***Barramento*** (*bus*)
  - ***conjunto de fios*** com uma função única (*vector*)
  - Ex: barramento de endereços (*address bus*),
  - Ex: barramento de dados (*data bus*)

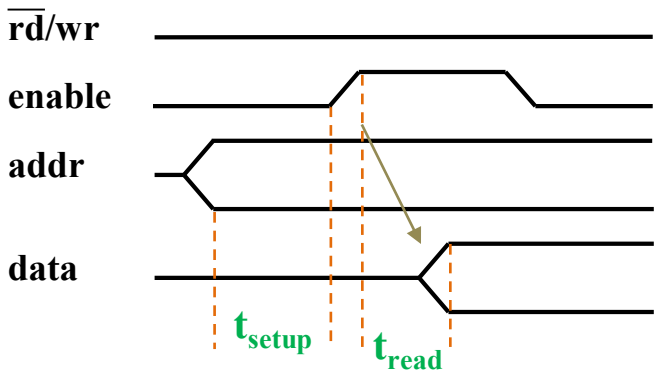
  - também, pode significar uma ***coleção de fios***
    - endereço, dado e controle,
    - ***protocolo*** agregado (com as regras de comunicação)



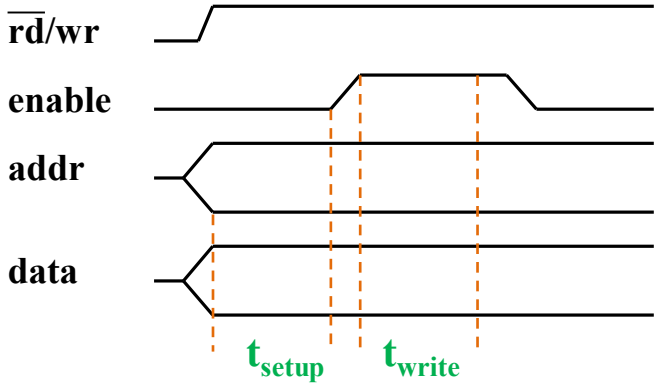
estrutura do barramento

➢ Diagrama de Tempos

- Método mais comum para descrever um protocolo de comunicação
- tempo incrementa para a direita no eixo x
- Sinais de controle: (*níveis low ou high*)
  - ¤ descrição de sinais ativos *low* ($\overline{rd}$, rd', /rd, rd_L)
  - ¤ usa-se termos ***assert*** (ativo) ou ***deassert*** (inativo)
  - ¤ e.g. *asserting* $\overline{rd}$ significa baixar o nível do sinal $\overline{rd}$.
- Sinais de dados: válidos ou não-válidos
- Protocolos podem ter subprotocolos
  - Chamar um *bus cycle* (para ler ou escrever)…
  - Cada subprotocolo pode usar vários ciclos de clock



read protocol



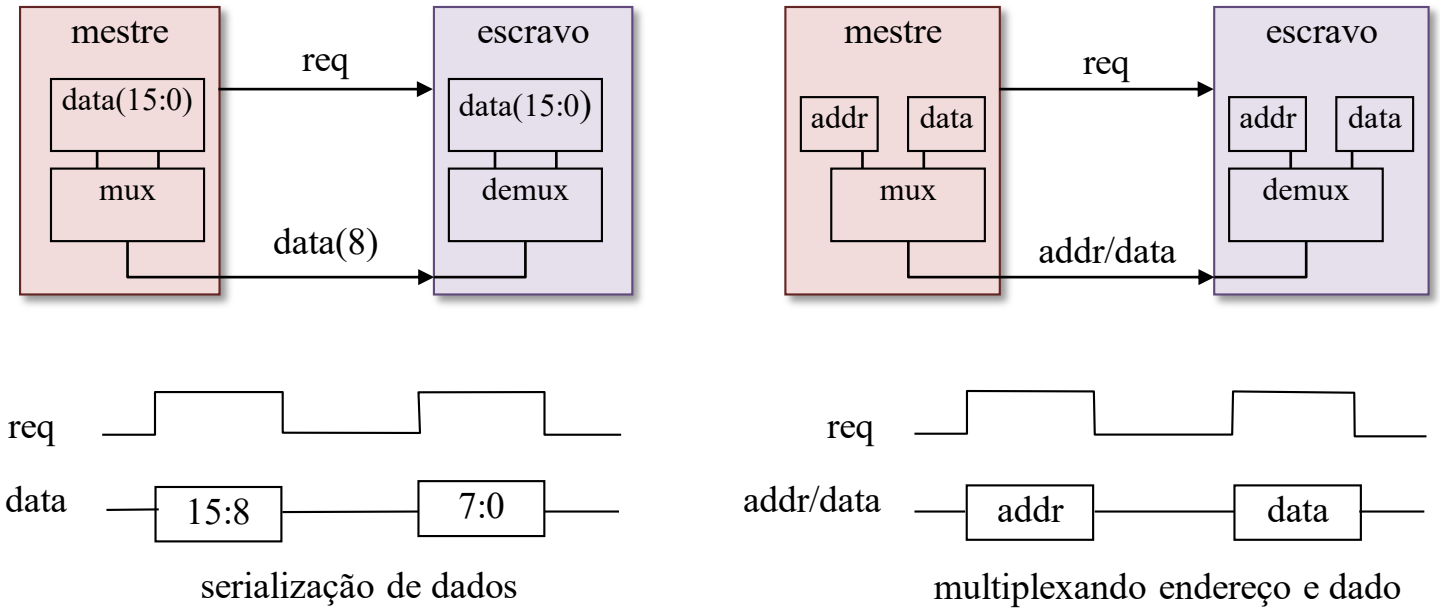
write protocol

Exemplo de leitura (*Read*)

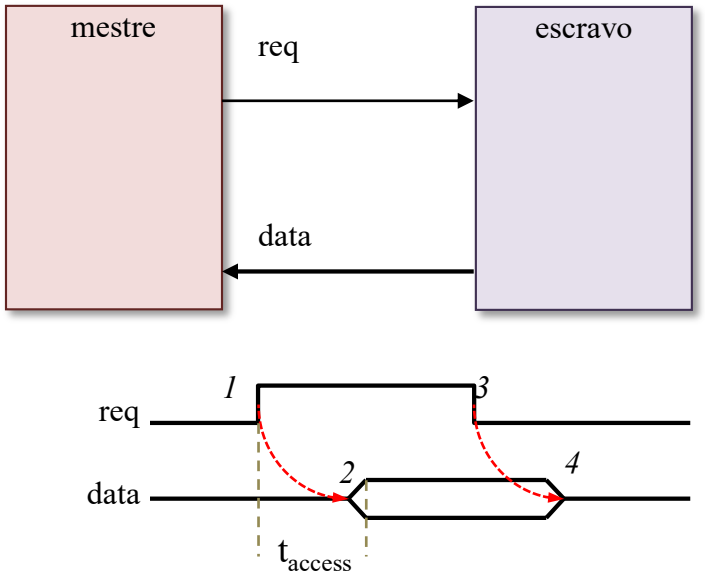
- ***rd/wr*** set low, address placed on ***addr*** for at least $\mathbf{t_{setup}}$ time before ***enable*** asserted, enable triggers memory to place data on ***data*** wires by time $\mathbf{t_{read}}$

- **Atores**: mestre (*master*) inicia, e o escravo (*slave*) responde
- **Direção**: de um emissor (*sender*) para um receptor (*receiver*)
- **Endereços**: tipo especial de dados
  - Especifica uma localização na memória, um periférico, ou um registrador específico num periférico;
- **Multiplexação temporal** (time multiplexing)
  - Compartilha um conjunto de fios para transportar múltiplos blocos de dados
  - Economiza fios ao custo de gastar tempo

Transferência de dados multiplexados temporalmente

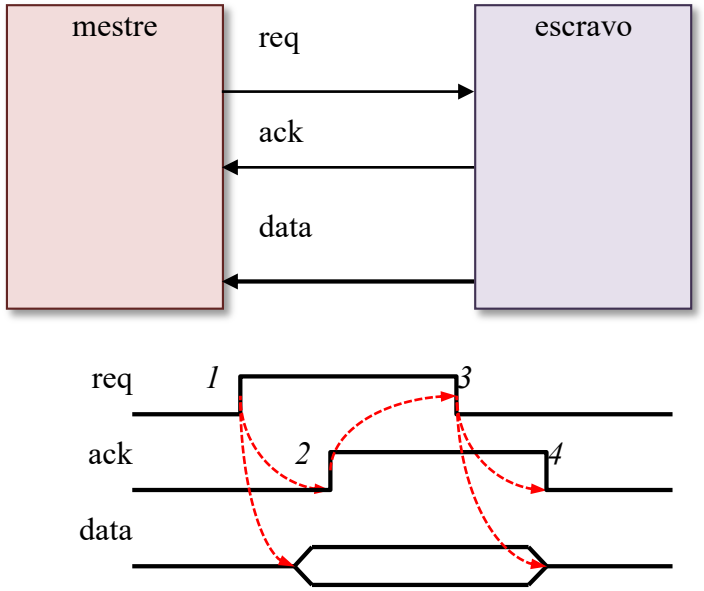


➢ Métodos de controle em protocolos



1. Mestre ativa (*asserts*) *req* para receber dados;
2. Escravo coloca o dado no *bus* dentro de um tempo $\mathbf{t_{access}}$
3. Mestre recebe o dado e desativa (*deasserts*) o sinal *req*
4. Escravo se torna pronto (ready) para próxima requisição

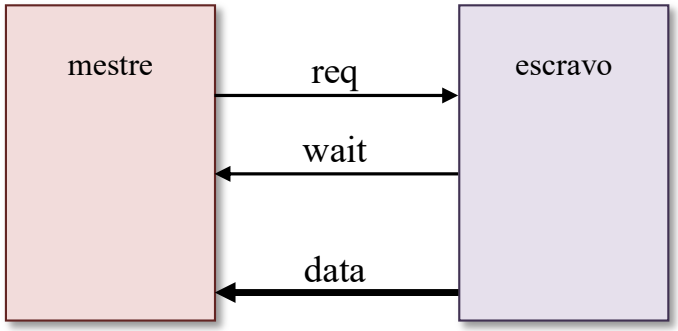
Protocolo STROBE (strobe protocol)



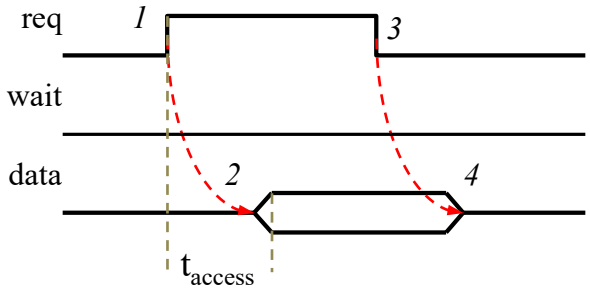
1. Mestre ativa *req* para receber dados
2. Escravo coloca dado no barramentoe E ativa ***ack***
3. Mestre recebe dado e desativa *req*
4. Escravo se torna pronto para uma próxima requisição

Protocolo HANDSHAKE (handshake protocol)

➢ Um protocolo STROBE / HANDSHAKE compromissado:

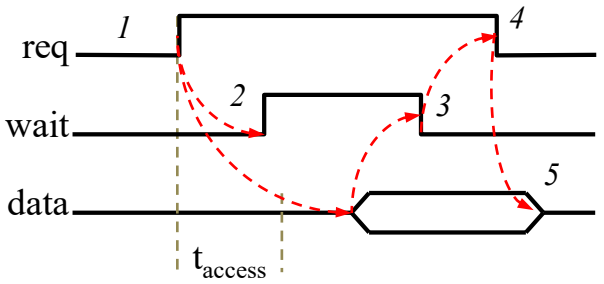


O compromisso é atender a requisição no tempo estipulado. Não sendo possível indica-se por um sinal **WAIT**, que o mestre respeita por algum tempo.



1. Mestre ativa *req* para receber dados
2. Escravo coloca dado no bus **dentro do tempo t_{access}** (linha de wait não é utilizada)
3. Mestre recebe o dado e desativa *req*
4. Escravo torna-se pronto para próxima requisição

Caso de resposta rápida

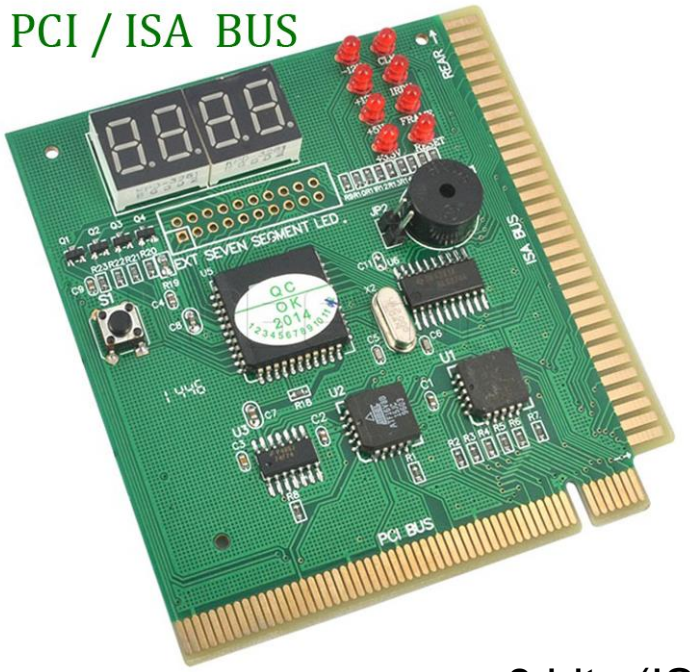


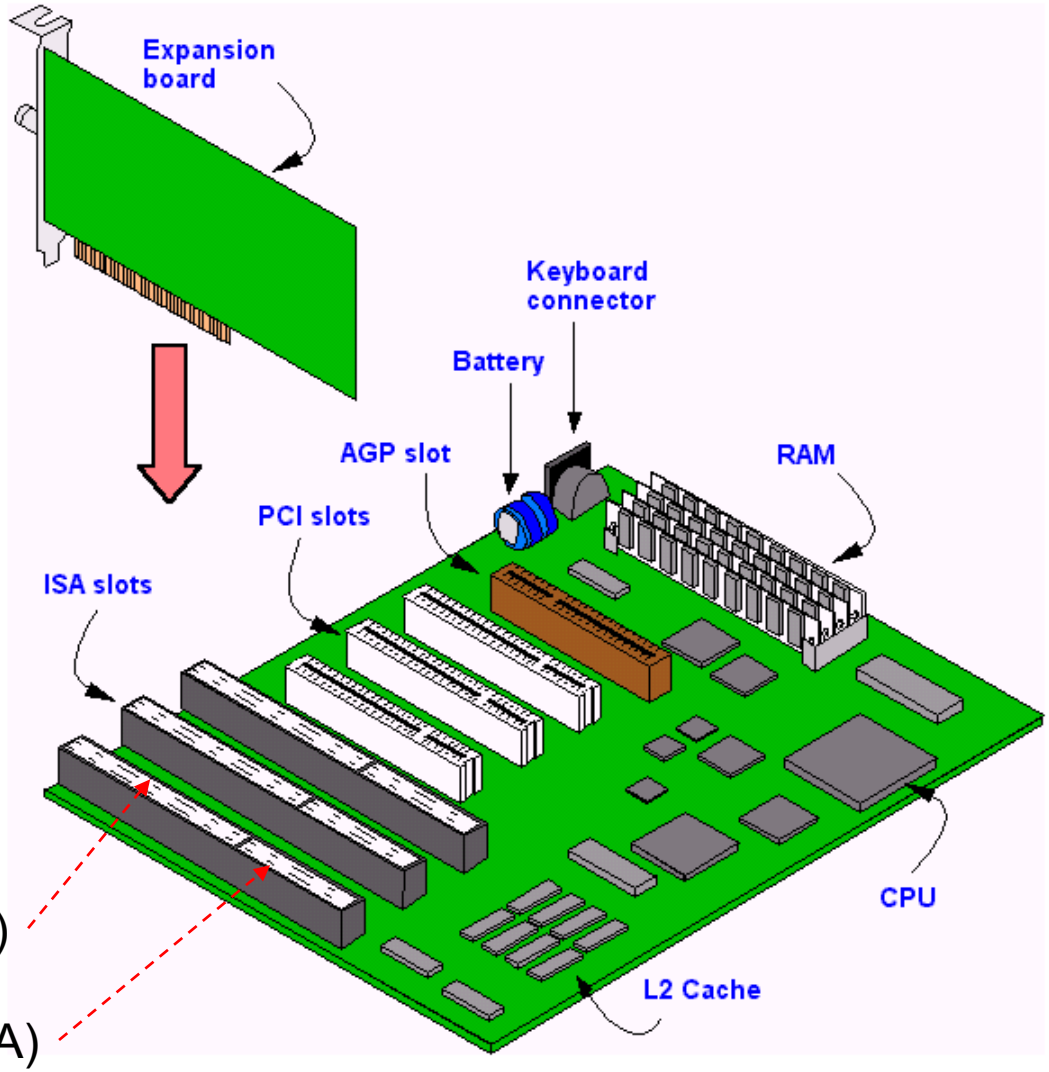
1. Mestre ativa *req* para receber dados
2. Escravo não pode colocar dados dentro do t_{access}, então, **ativa *wait***
3. Escravo coloca dados no bus e **desativa *wait***
4. Mestre recebe o dado e desativa *req*
5. Escravo torna-se pronto para próxima requisição

Caso de resposta lenta (com atraso – delay)

O barramento para computadores ISA (acrônimo para ***Industry Standard Architecture***) foi padronizado em 1981, inicialmente utilizando *8 bits (IBM-PC/XT)* para a comunicação entre CPU e periféricos, e posteriormente adaptado para *16 bits* (PC-AT Intel 80286).

A concepção de barramento de interface...

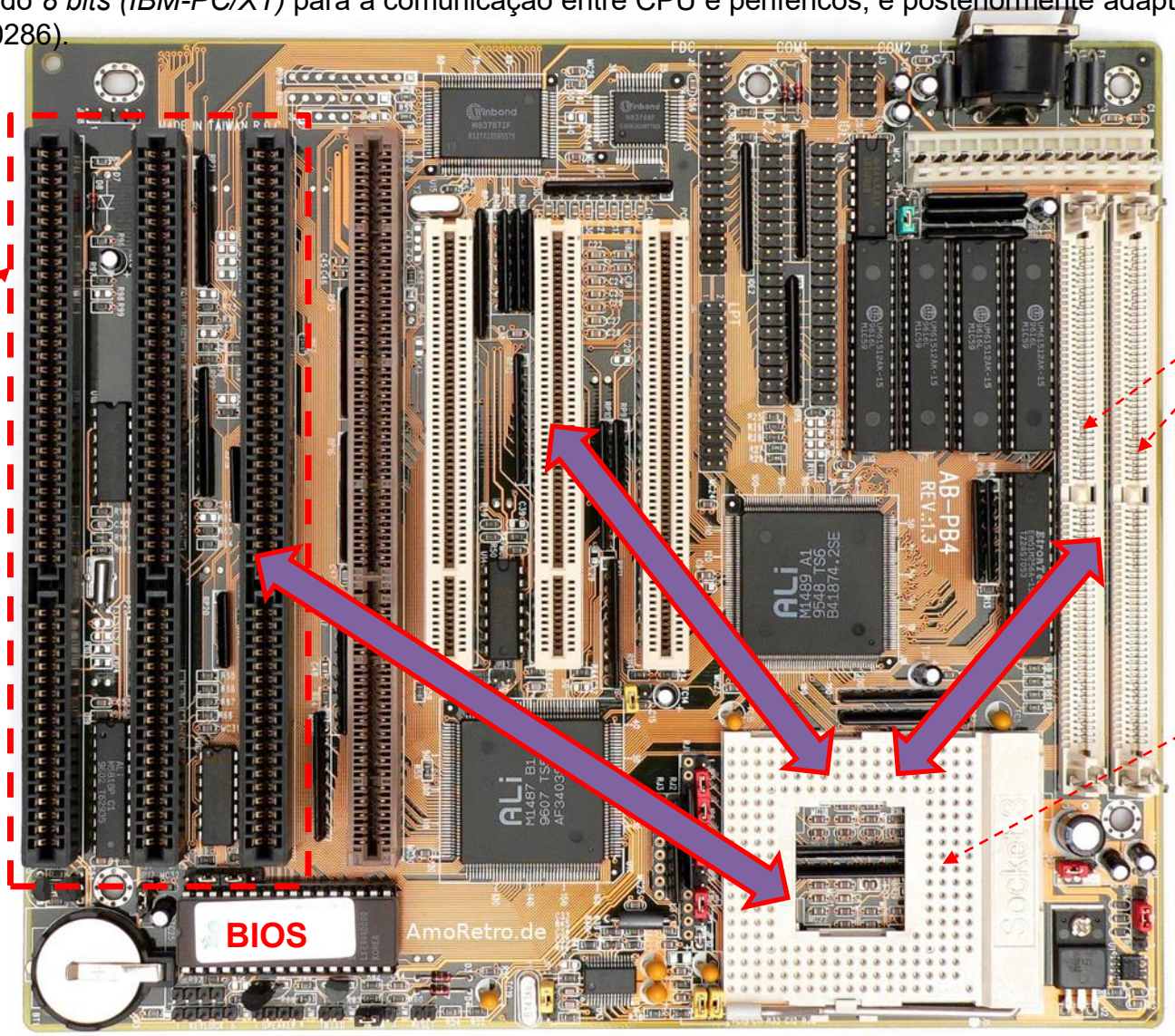


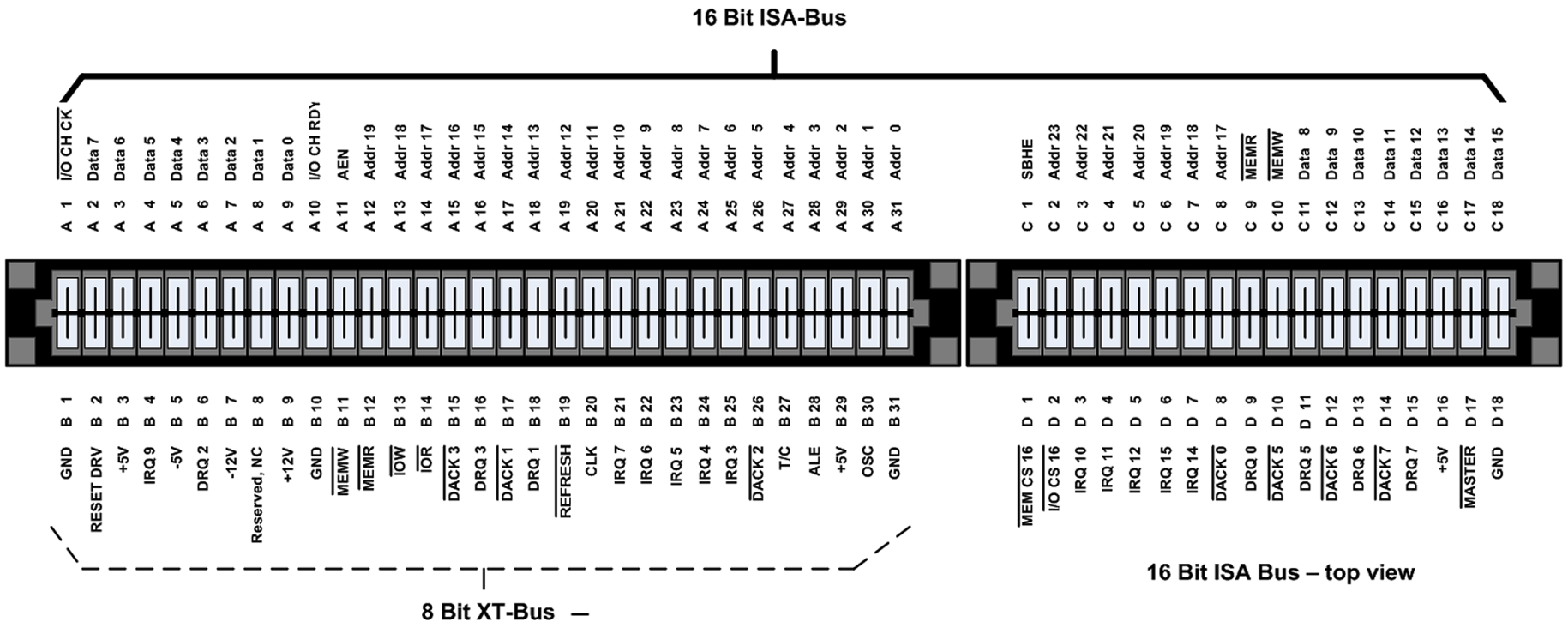


8 bits (ISA)

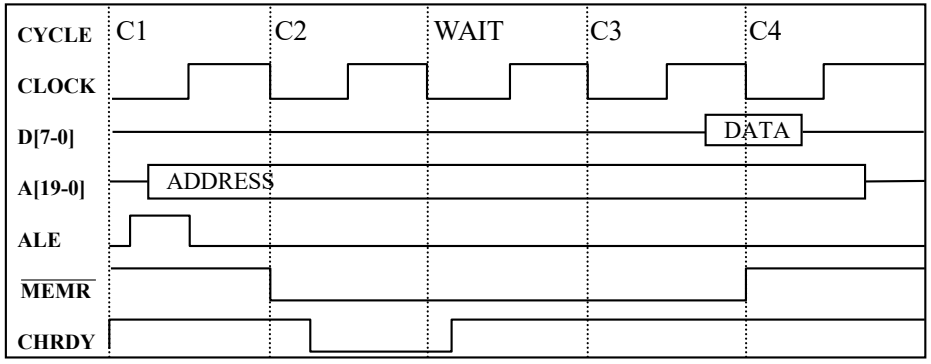
16 bits (EISA)

O barramento para computadores ISA (acrônimo para ***Industry Standard Architecture***) foi padronizado em 1981, inicialmente utilizando *8 bits (IBM-PC/XT)* para a comunicação entre CPU e periféricos, e posteriormente adaptado para *16 bits* (PC-AT Intel 80286).

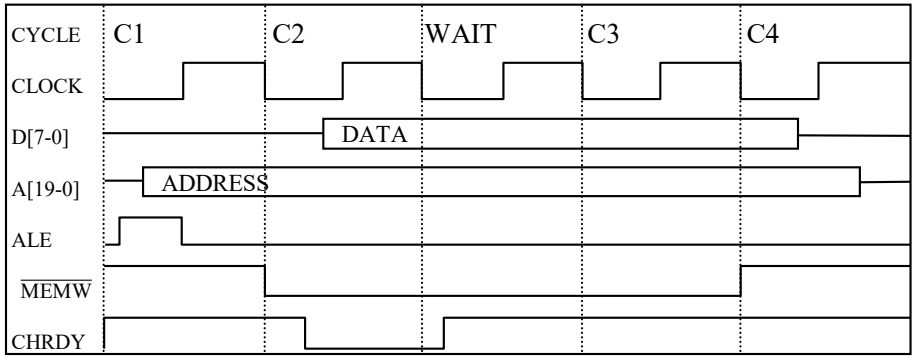


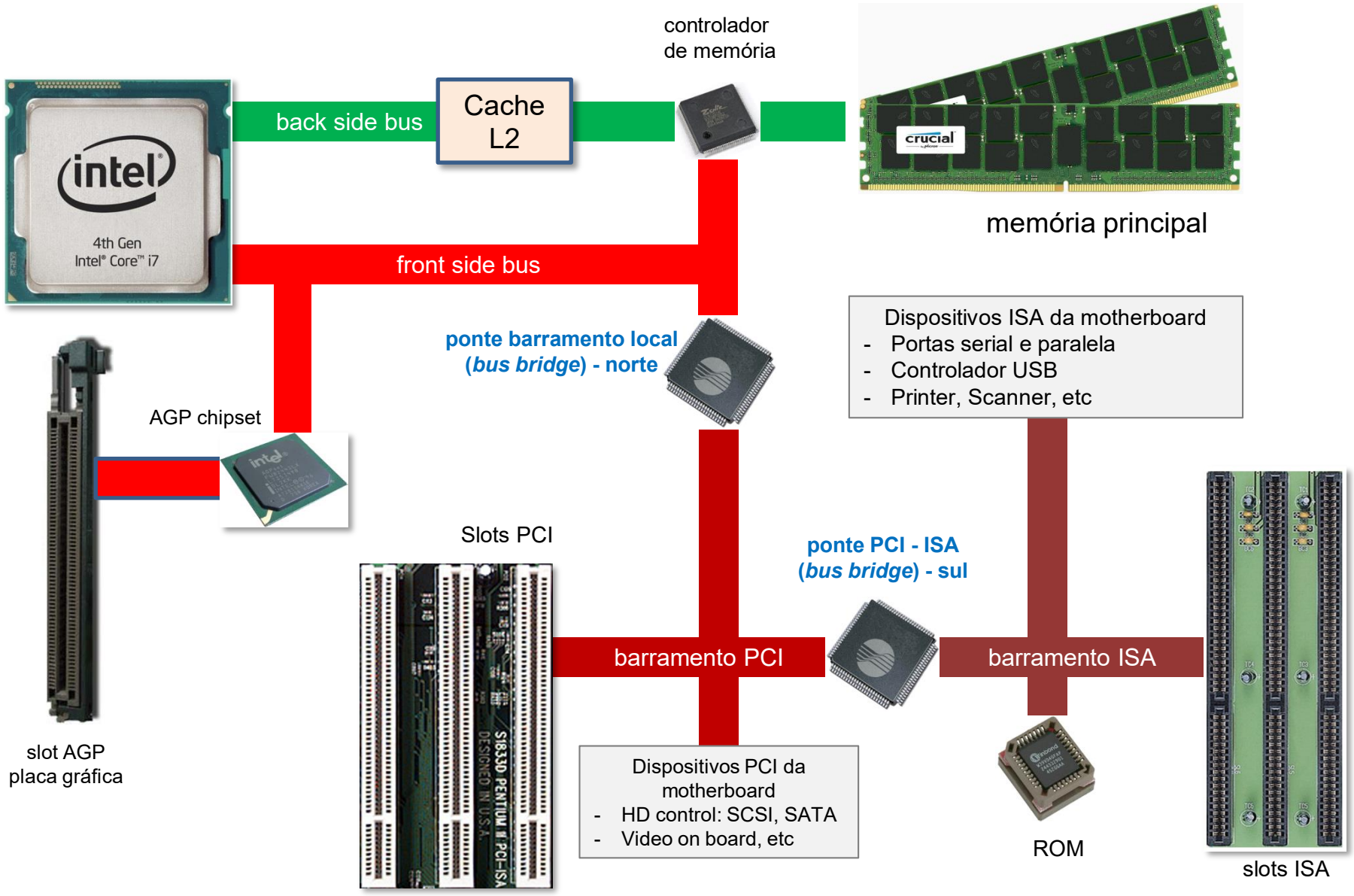


memory-read bus cycle

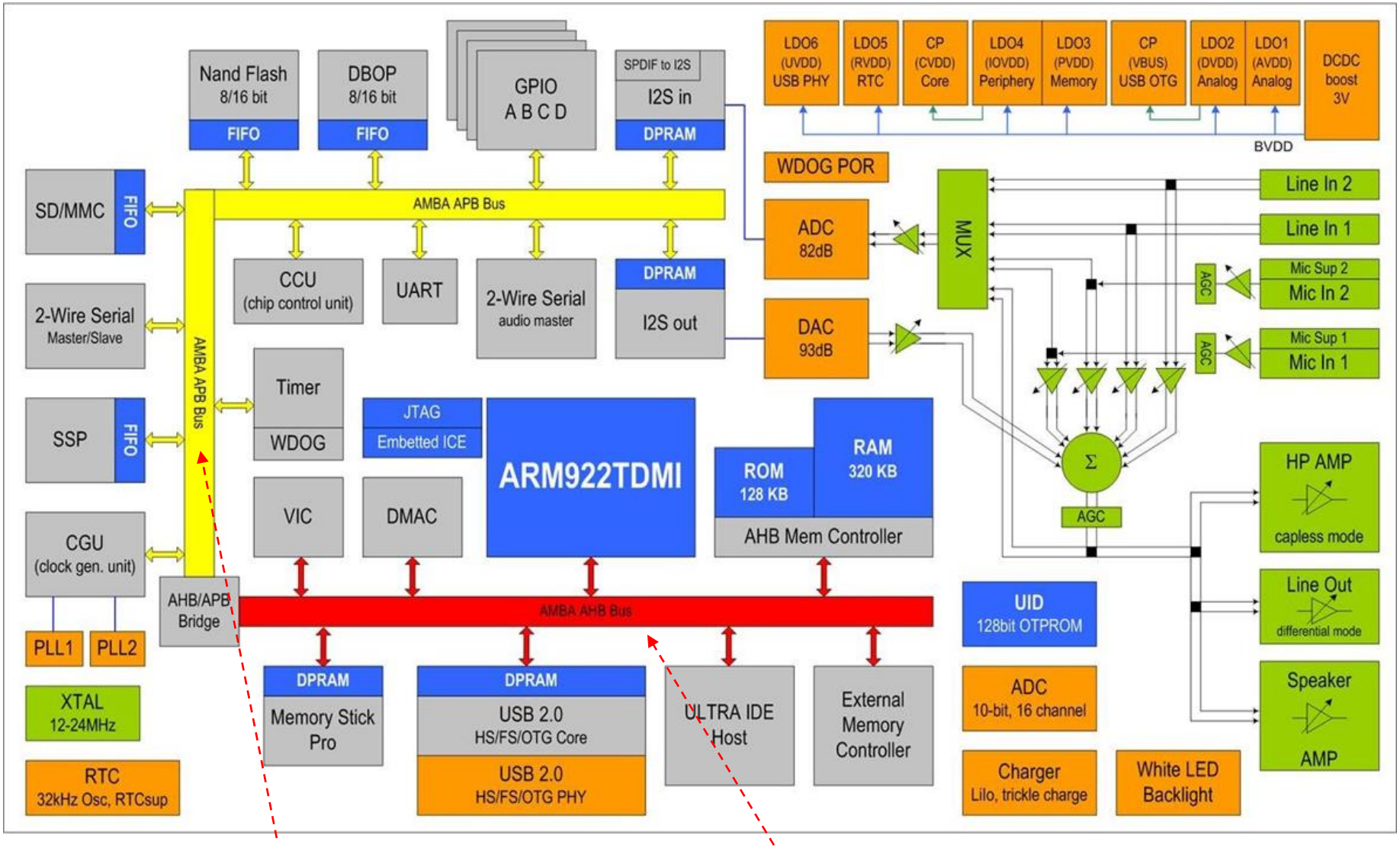


memory-write bus cycle



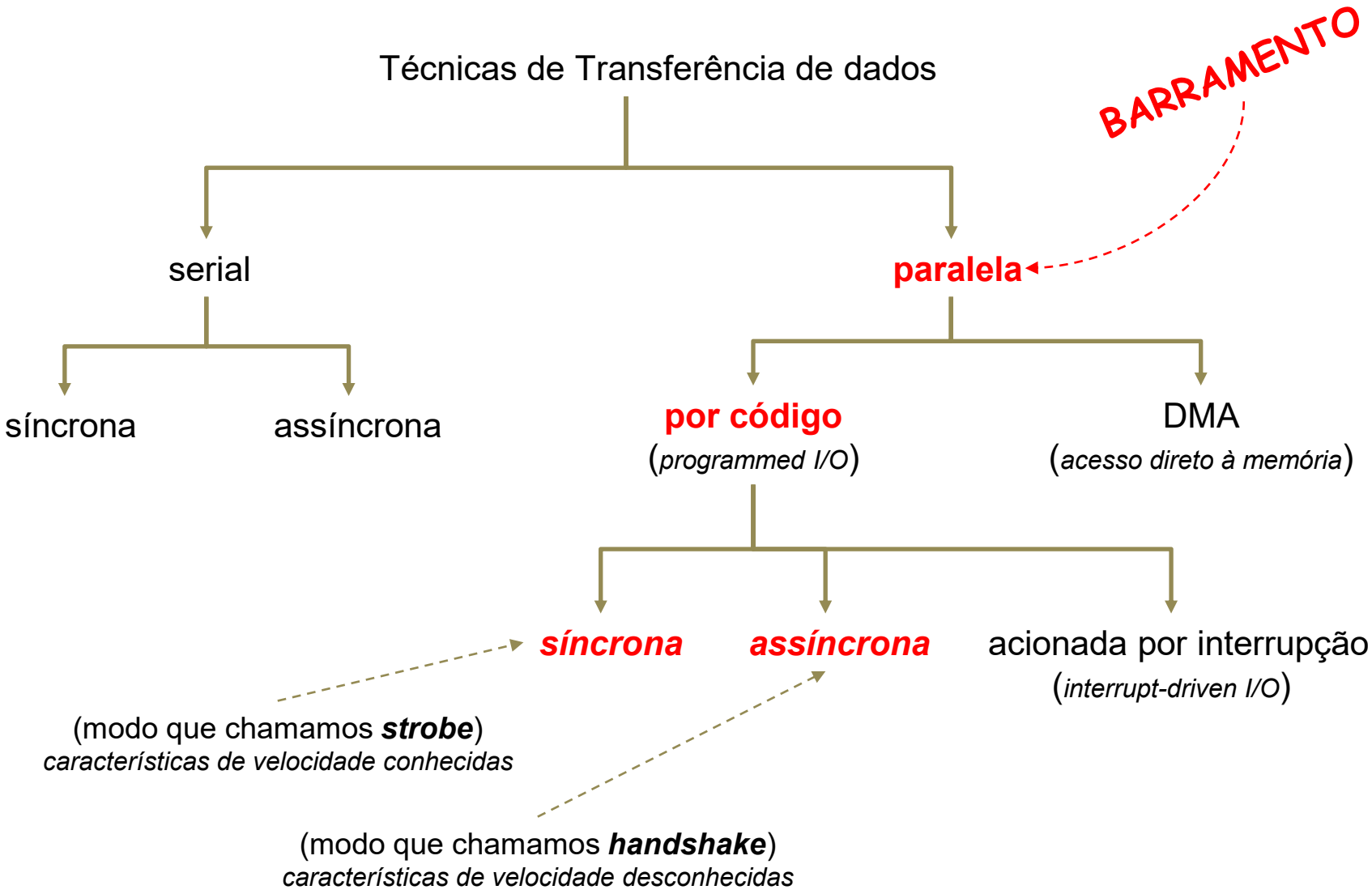

controlador
de memória
back side bus
Cache
L2
memória principal
front side bus
ponte barramento local
(bus bridge) - norte
Dispositivos ISA da motherboard
- Portas serial e paralela
- Controlador USB
- Printer, Scanner, etc
AGP chipset
slot AGP
placa gráfica
Slots PCI
ponte PCI - ISA
(bus bridge) - sul
barramento PCI
barramento ISA
Dispositivos PCI da
motherboard
- HD control: SCSI, SATA
- Video on board, etc
ROM
slots ISA

A ARM desenvolveu um barramento próprio (AMBA) – versão atual: AXI (*exemplo de um SoC*)

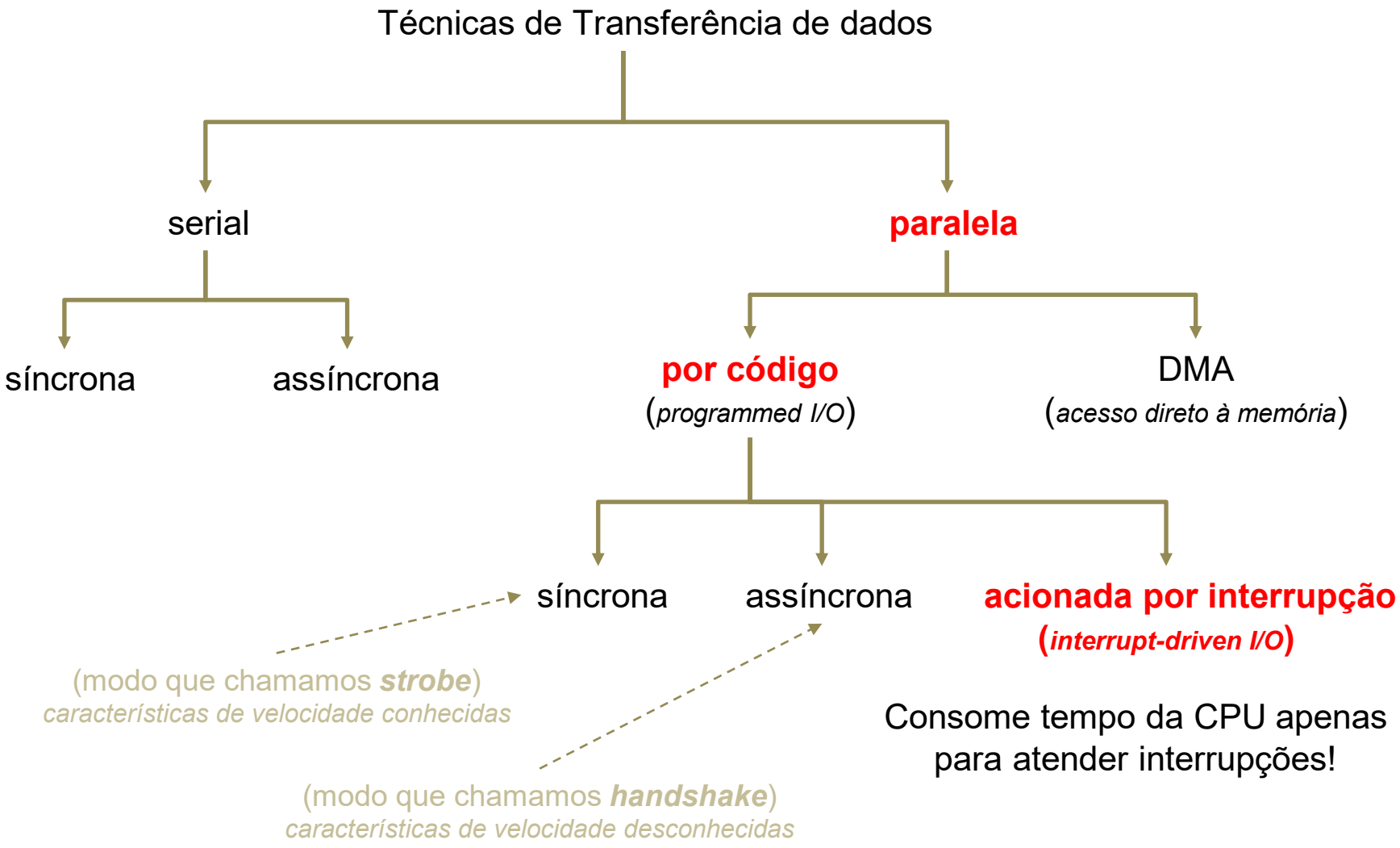


APB = advanced periferal bus (mais lento)

AHB = advanced high-performance bus



Ambos consomem tempo da CPU...


Técnicas de Transferência de dados
serial
paralela
síncrona
assíncrona
por código
(programmed I/O)
DMA
(acesso direto à memória)
síncrona
assíncrona
acionada por interrupção
(interrupt-driven I/O)
(modo que chamamos strobe)
características de velocidade conhecidas
(modo que chamamos handshake)
características de velocidade desconhecidas
Consome tempo da CPU apenas
para atender interrupções!

- Vários periféricos geram dados de forma intermitente que devem ser processados. – atendidos pela CPU, o que é chamado "***servicing***". Se o periférico gera dados em intervalos não regulares, como o μP sabe o momento de atender um serviço desse periférico?

- *Polling* (varredura) – a CPU verifica a existência de dados novos desse periférico, isso pode gastar tempo da CPU e fica inviável com muitos periféricos, ou quando o periférico requer atendimento urgente.

- *Interrupção* – o periférico externamente comunica ao μP que requer atenção, e a CPU interrompe o processo em andamento para atender à interrupção.

  → O μP geralmente tem alguns pinos, e.g. $\overline{\text{INT}}$, que ao final de cada instrução é verificado. Se estiver ativo (*asserted*) o μP faz um *jump* para uma área onde se encontra uma *subrotina* chamada ***Interrupt Service Routine (ISR)***. Os periféricos que usam esse recurso são chamados ***interrupt-driven I/O***.

  → Há 2 métodos para o μP saber qual ISR executar para atender a um periférico que o interrompeu:

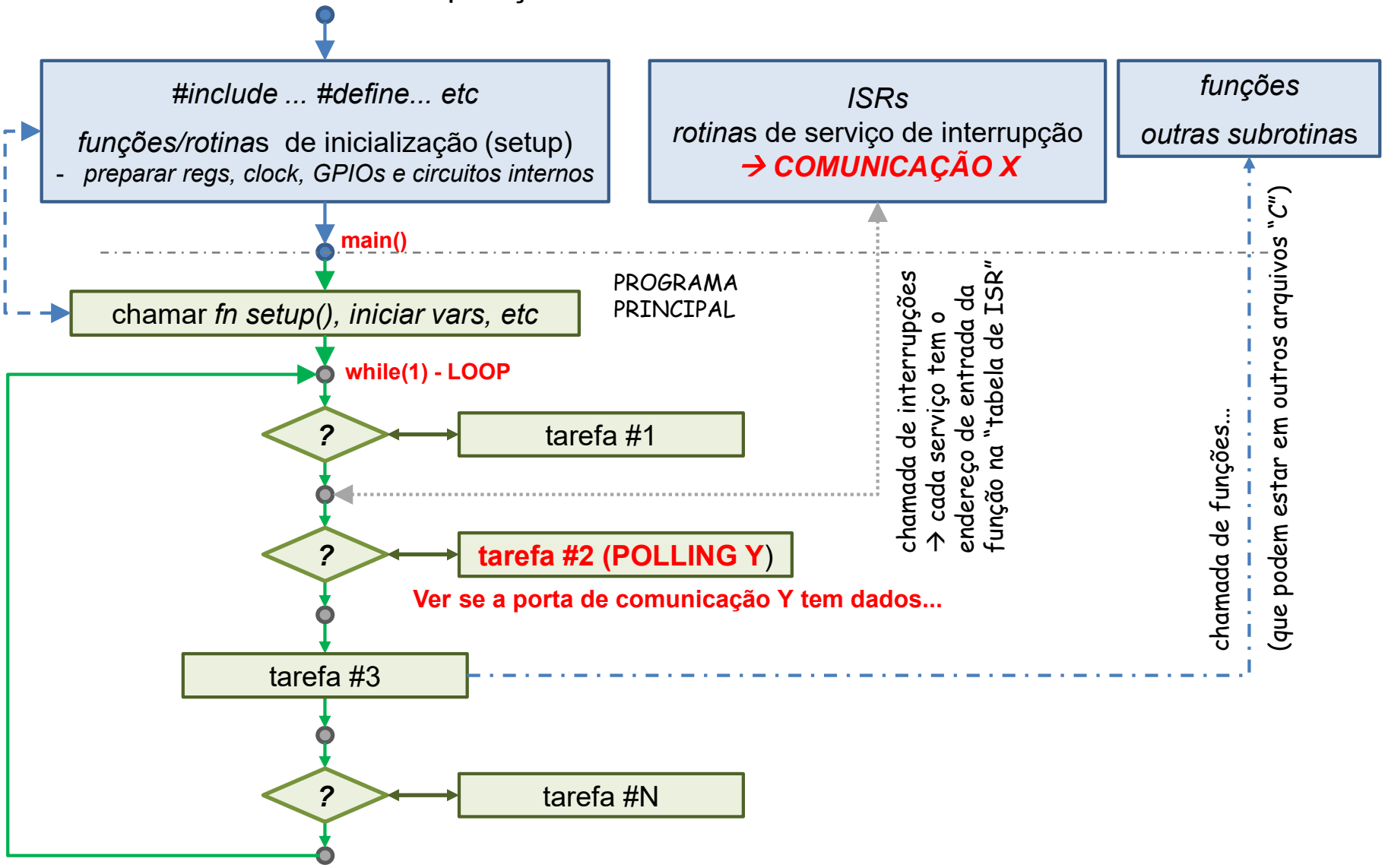
  ❖ Fixo (***fixed interrupt***) – o endereço onde a ISR deve ser escrita é predefinido na construção do μP,

  ❖ Vetorial (***vectored interrupt***) – o periférico que pediu a interrupção deve colocar no data bus um vetor que o identifica.

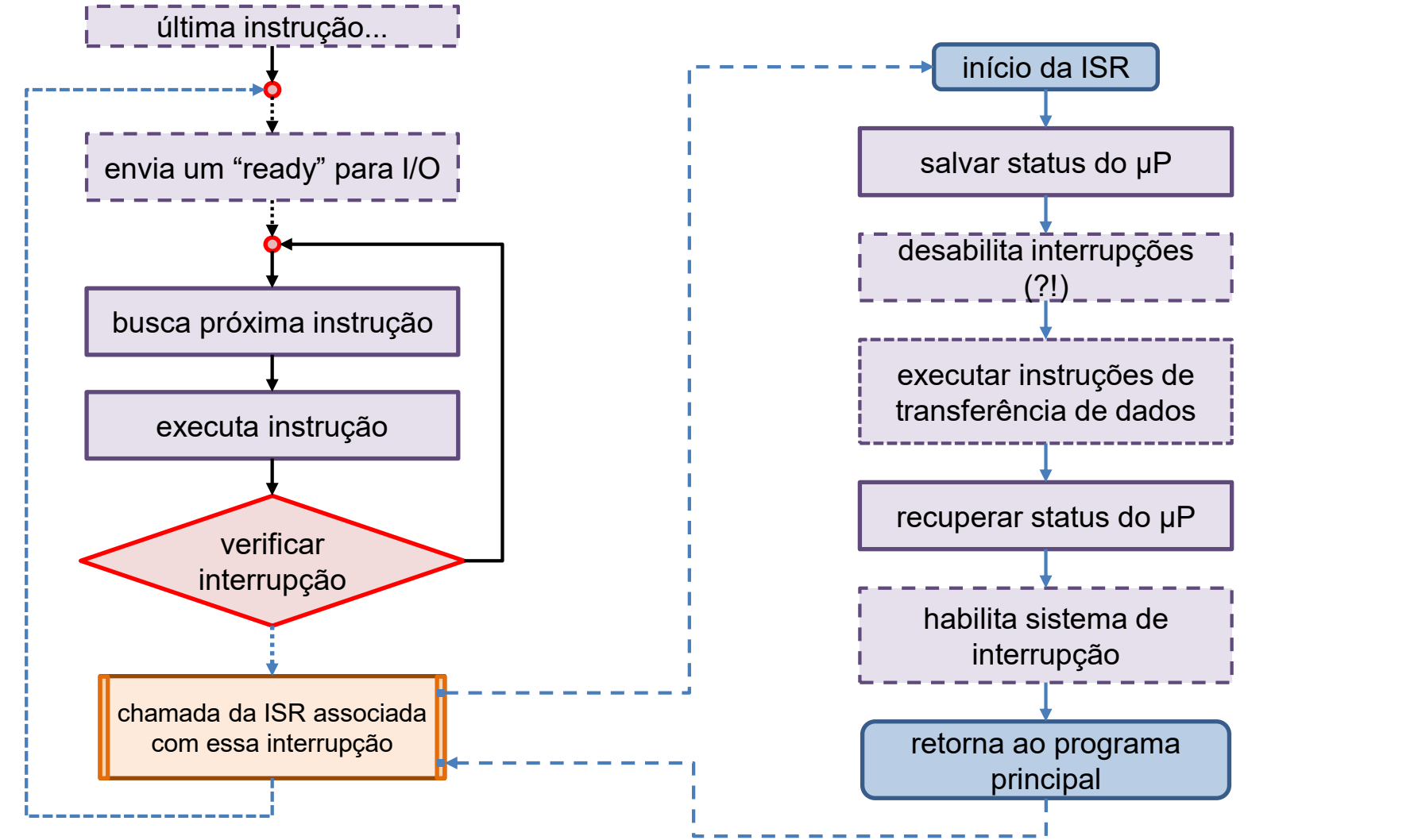
a) Periférico aciona $\overline{\text{INT}}$ e o μP checa ao final de cada instrução,
b) O μP aciona outro pino ($\overline{\text{INTA}}$) indicando que reconheceu o pedido de interrupção e vai acionar uma ISR (qual?)
c) O periférico então coloca no data bus um vetor indicando qual ISR será atendida (ou quem pediu a interrupção)
d) O μP se baseia em uma tabela (***interrupt address table*** – associa vetor e endereço da ISR) para saltar para ISR adequada.

(***vantagem***: cada periférico tem seu apontador na tabela de ISR, e cada ISR pode ser localizada em qualquer área da memória)

➢ Reformulando o modo de operação da CPU... conceito de SUPERLOOP...

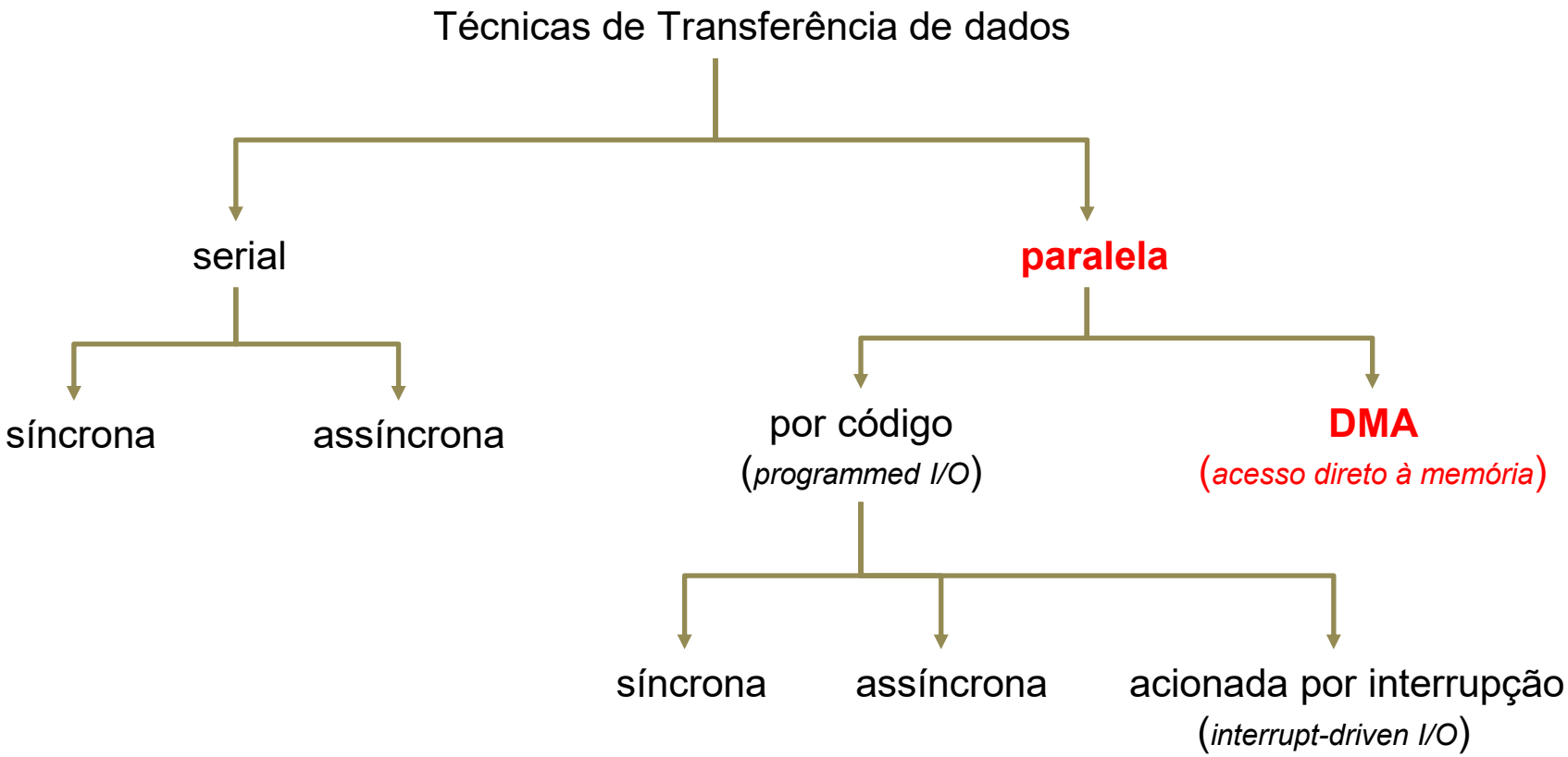


➢ Ciclo de instruções (*pipeline*) do μP *sem* e *com* pedido de interrupção...

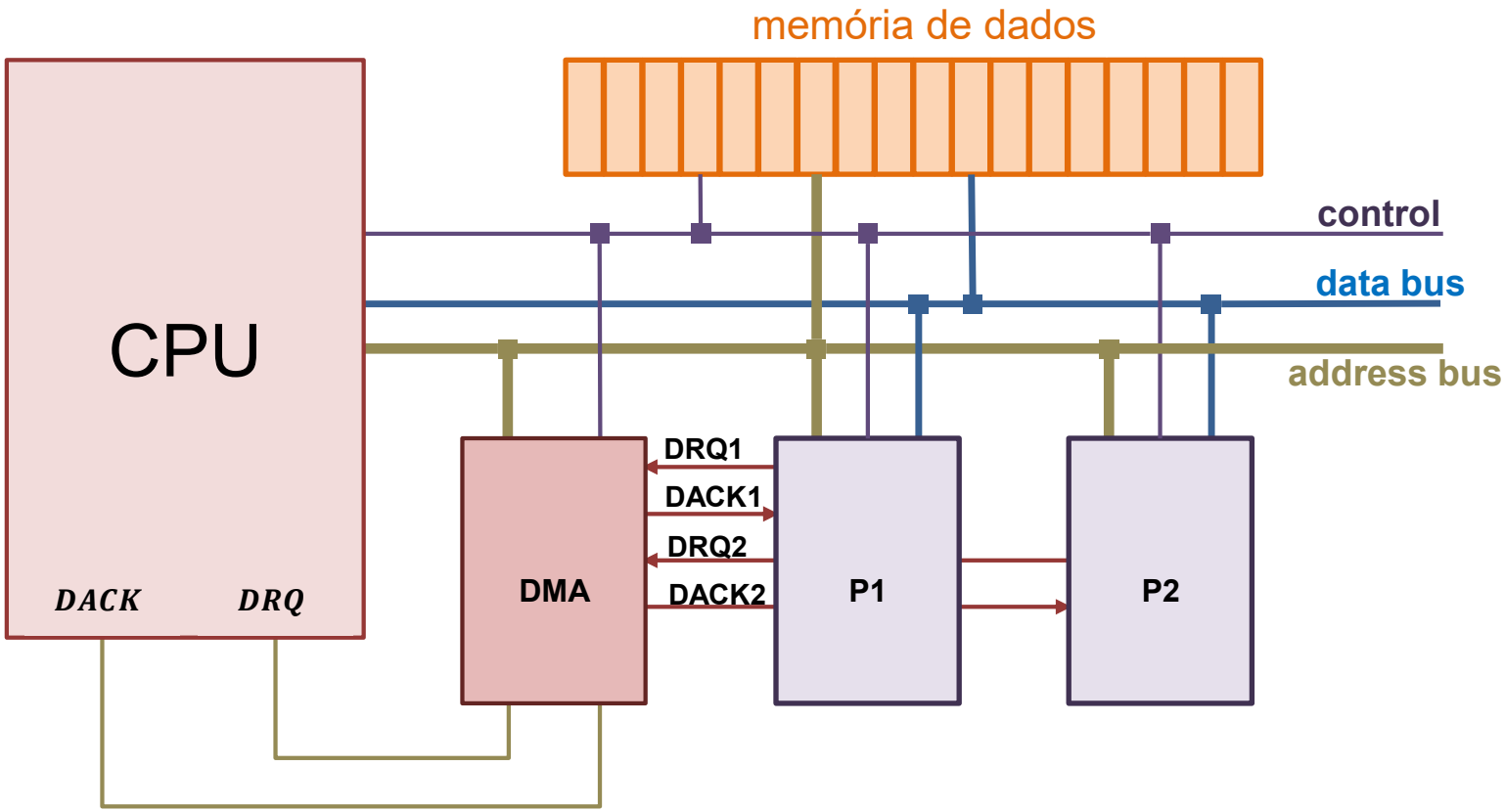


sequência de execução do programa principal...

sequência de execução da rotina de serviço da interrupção (ISR)


Técnicas de Transferência de dados
serial
paralela
síncrona
assíncrona
por código
(programmed I/O)
DMA
(acesso direto à memória)
síncrona
assíncrona
acionada por interrupção
(interrupt-driven I/O)

- Vários periféricos produzem dados em grande quantidade
- Usar a CPU para transferir cada bloco de dados para a RAM pode ser muito lento...
- Melhor seria usar um processador dedicado, especializado em transferir dados rapidamente;
- Esse processador se chama **Direct Memory Access** controller (DMA)



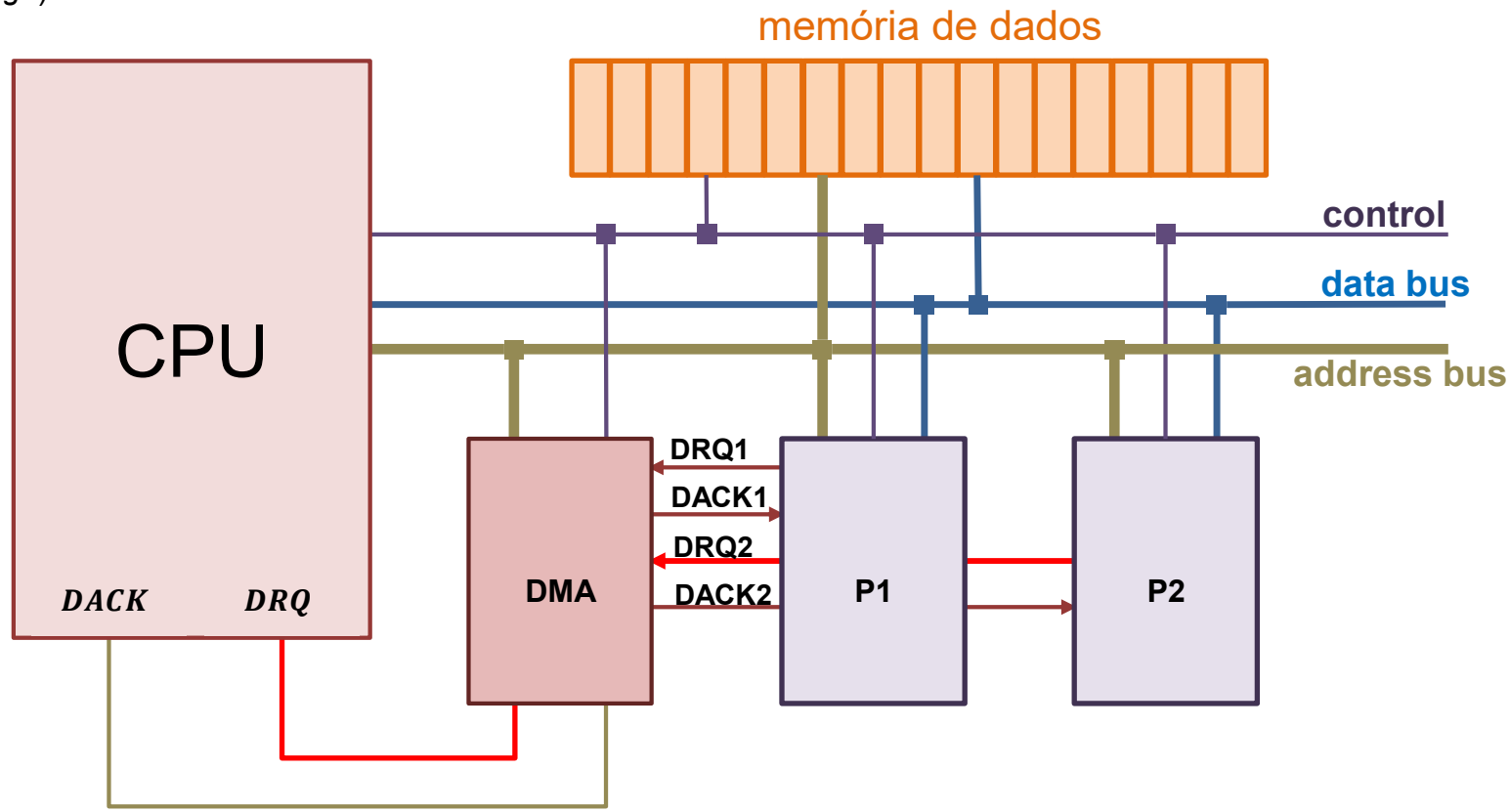
➢ Como funciona:

a) Para cada periférico, a CPU primeiro ***configura*** o DMAc com pelo menos 3 parâmetros:

- Endereço do periférico de onde os dados devem ser lidos / escritos
- Endereço da memória onde os dados devem ser escritos / lidos
- A quantidade total de bytes que será transferida na operação.

b) Periféricos geralmente têm buffers de memória; quando cheio, o periférico solicita serviço ao DMAc acionando ***DRQx***

c) O DMAc solicita controle do barramento ativando **DRQ** (*data request*) da CPU... ao final da instrução atual, a CPU pode liberar o controle do *data bus*, *address bus*, e os sinais de *controle*. Para avisar ao DMAc a CPU ativa **DACK** (*data acknoledge*).



➢ Como funciona:

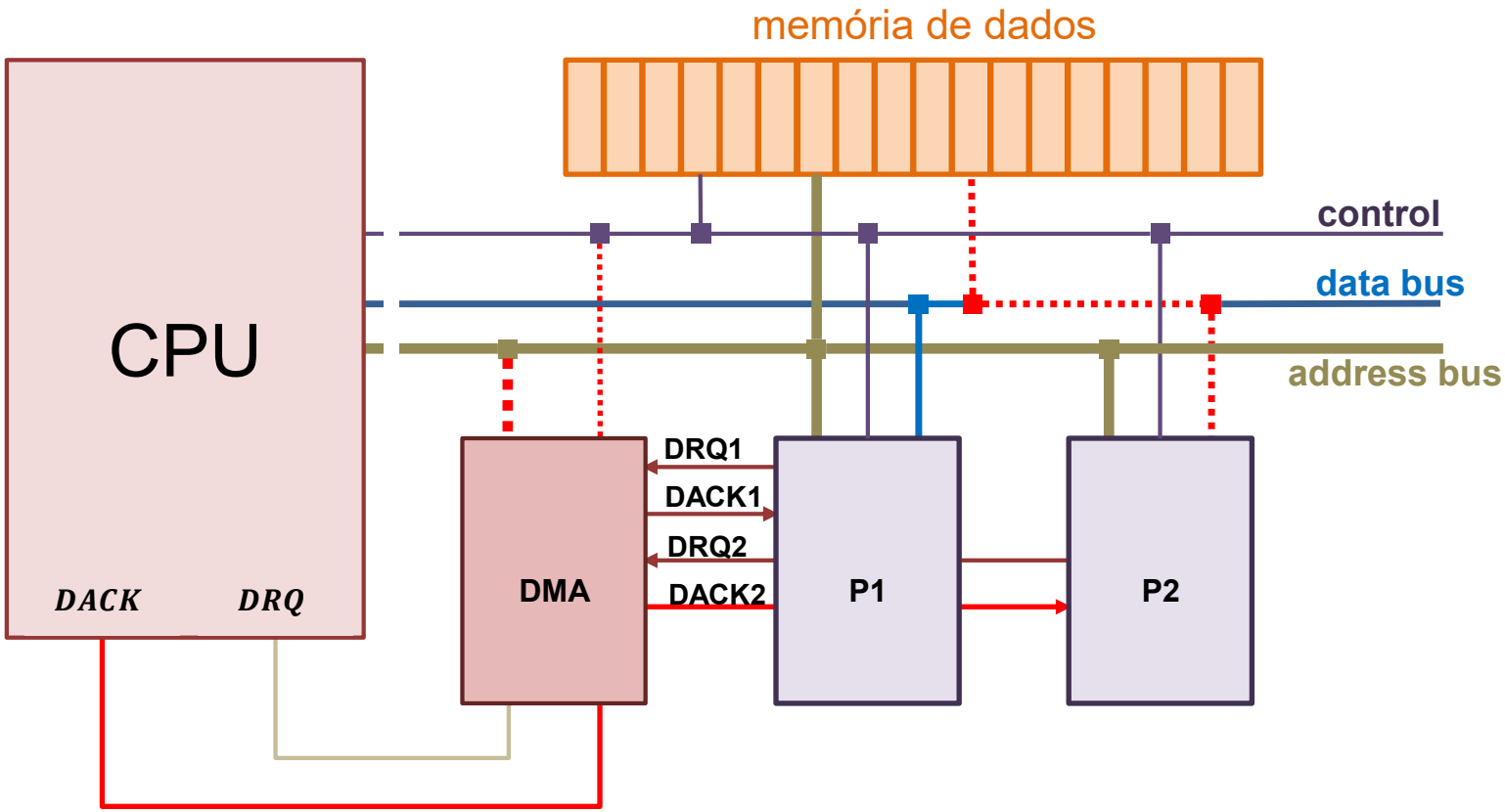
d) O DMAc então avisa o periférico que o serviço dele será atendido ativando DACKx (data acknoledge)

e) A CPU então coloca todos os sinais dos barramentos em 3-state (nesse momento a CPU pode executar instruções na cache interna)

f) O DMAc assume o controle do bus de endereços e os sinais de controle – os dados são fornecidos pelo periférico!

g) Quando chega ao final da transferência do bloco o DMAc desativa DACKx para o periférico,

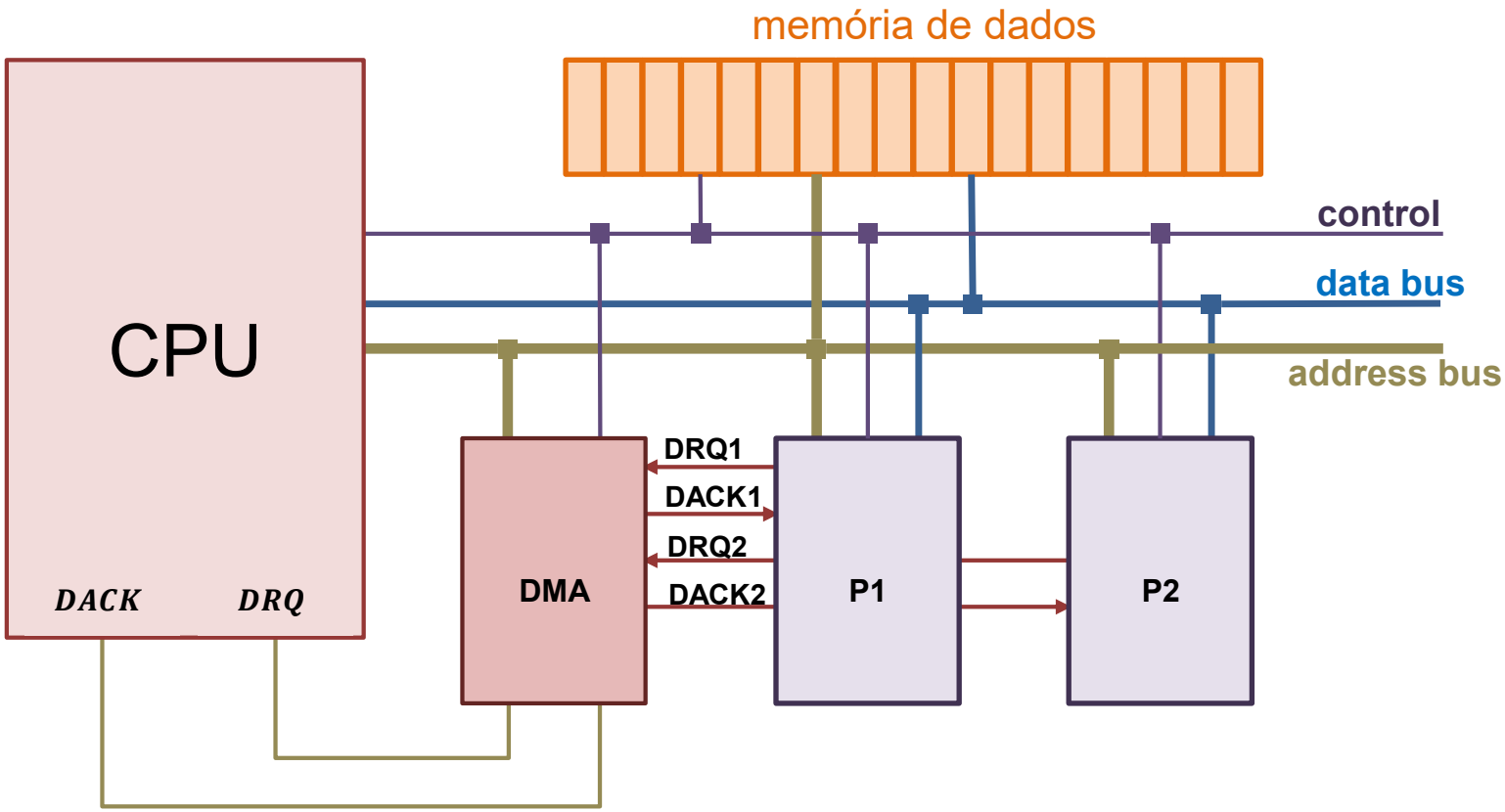
h) Pode ser que (padrões antigos) o DMAc peça interrupção à CPU para que ela organize (fique ciente) dados novos na memória.



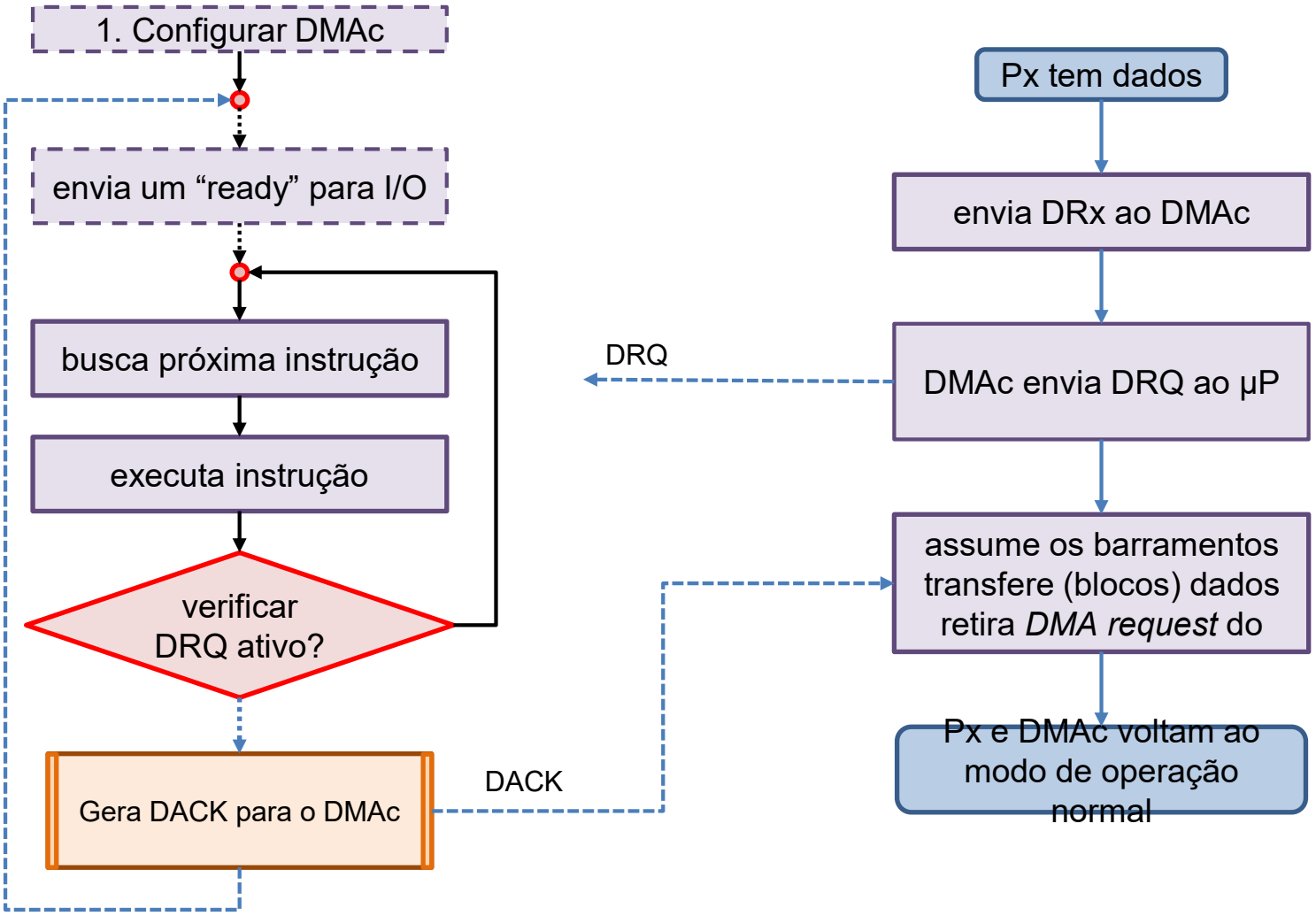
➢ Como funciona:

i) A CPU atende um ISR do DMA (ou não) e retoma o controle do barramento; assim, continua a execução do programa principal.

Note-se a transferência usando DMA pode ser de um único byte ou de um bloco (buffer) de memória.
Se a CPU estiver rodando o programa principal e acontecer um DMA (especialmente 1 palavra apenas) ela não precisa salvar conteúdo na pilha, nem recuperá-los ao retorno da interrupção. Isso aconteceria em cada palavra/byte transferido.
Se a CPU tem um cache primário (L1) e/ou um cache secundário razoável (L2) é possível que DMA aconteça em paralelo à execução do programa principal – Isso é especialmente verdade nos sistemas mais modernos...



➢ Ciclo de instruções (*pipeline*) do μP *com* um uma transferência em DMA...

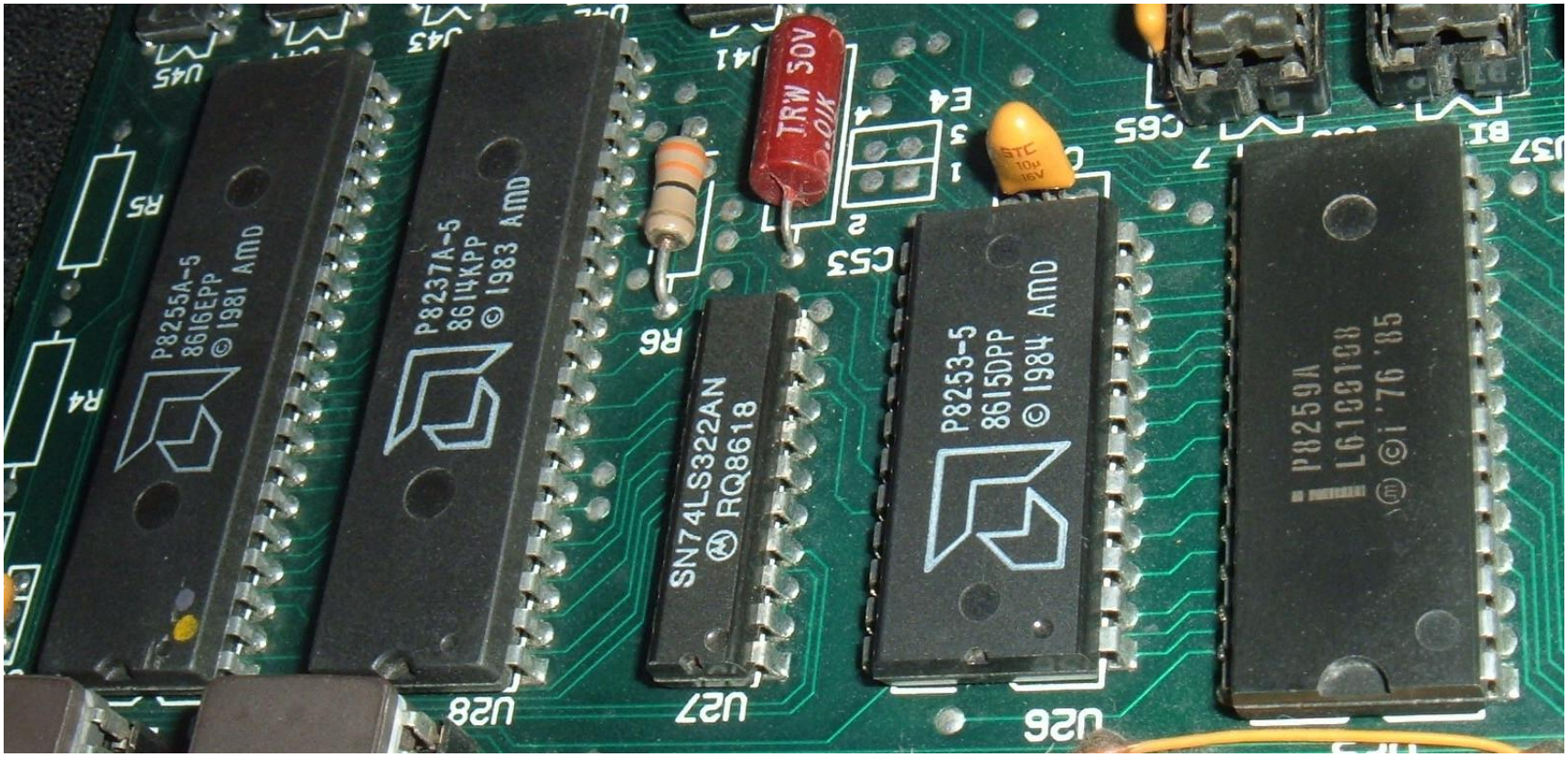


sequência de execução do programa principal...

sequência em paralelo da operação de DMA

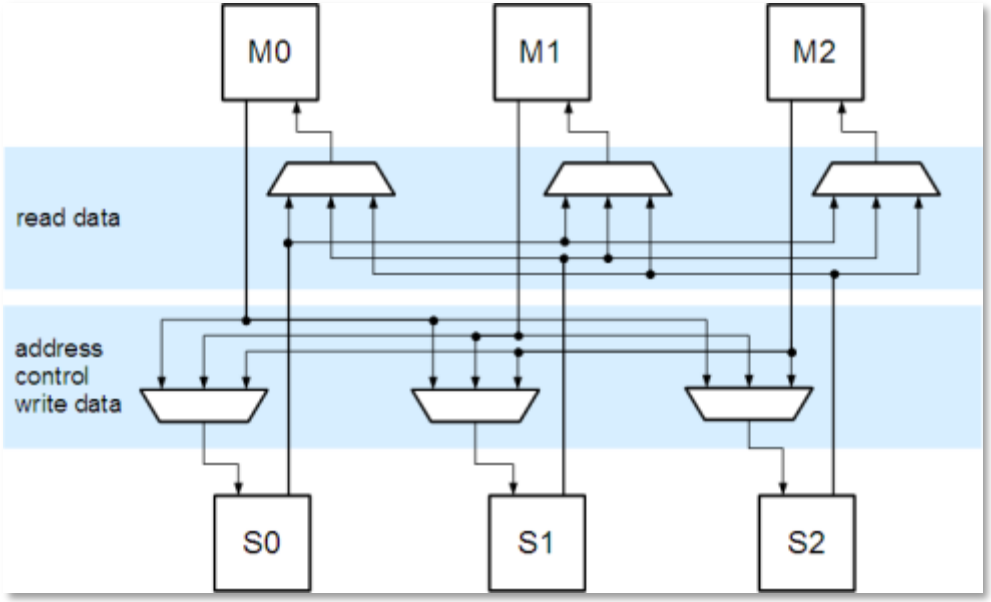
- Nos PCs mais antigos, esses costumavam ser os CIs que controlam interrupções, DMA e periféricos:

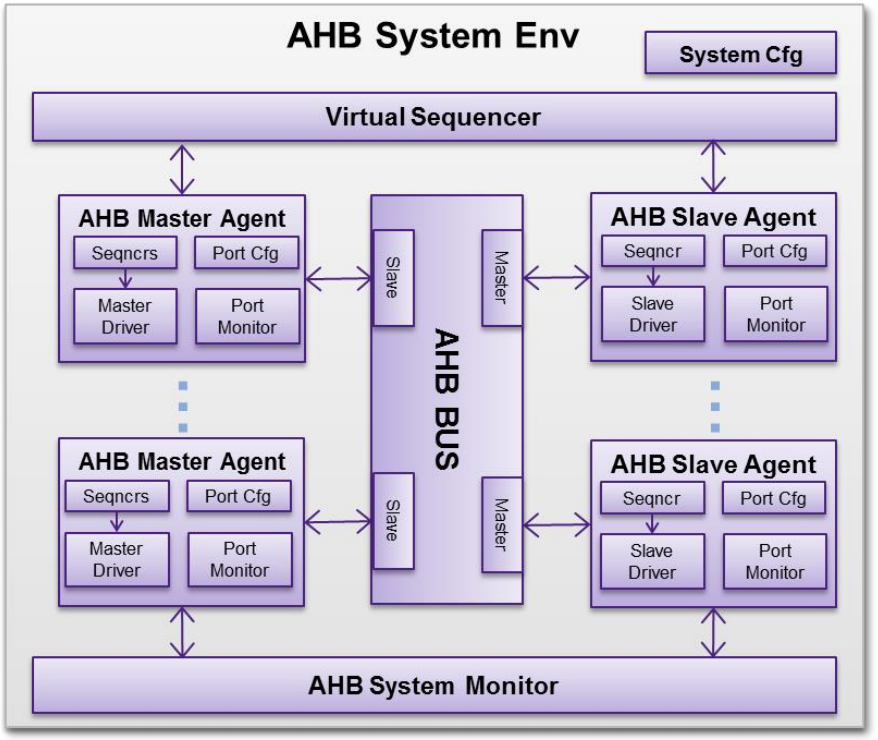
- programmable peripheral interface (**PPI**) P8255A
- direct memory access (**DMA**) controller P8237A,
- Programmable Interval Timers (PITs) P8253,
- Programmable Interrupt Controller (**PIC**) P9259A.



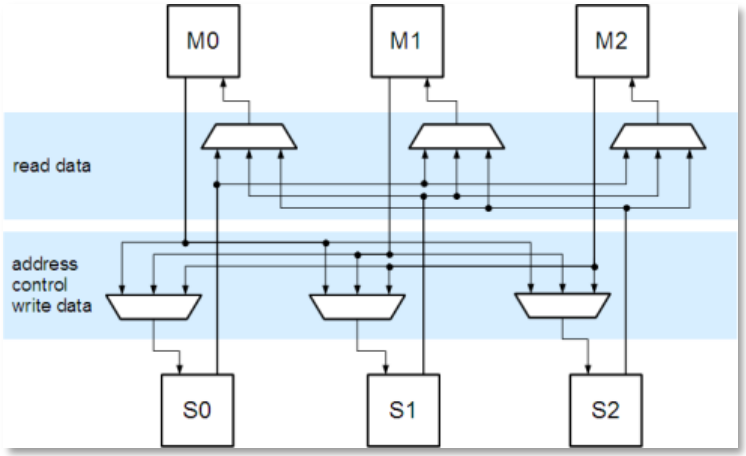
Atualmente todos esses CIs foram integrados nos μP, nos μC e SoC.

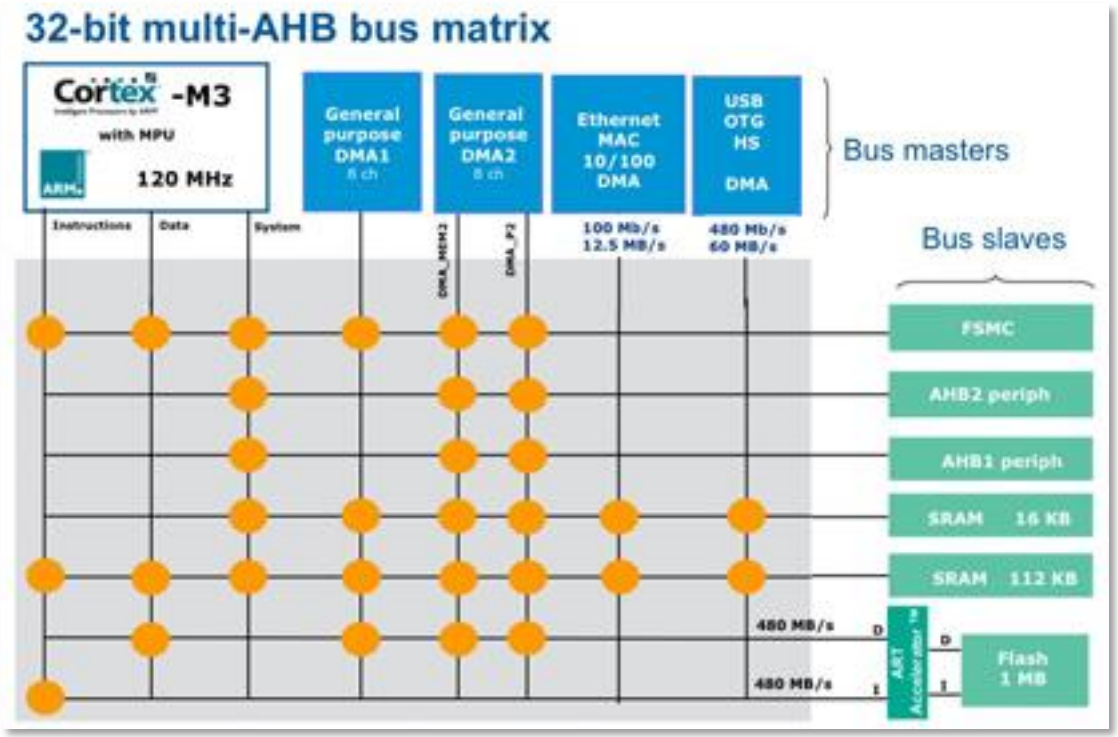
- The ARM **Advanced Microcontroller Bus Architecture (AMBA)**
- Conceito novo em DMA é **Multilayer Bus Matrix** – com vários canais de DMA, vários mestres e escravos!

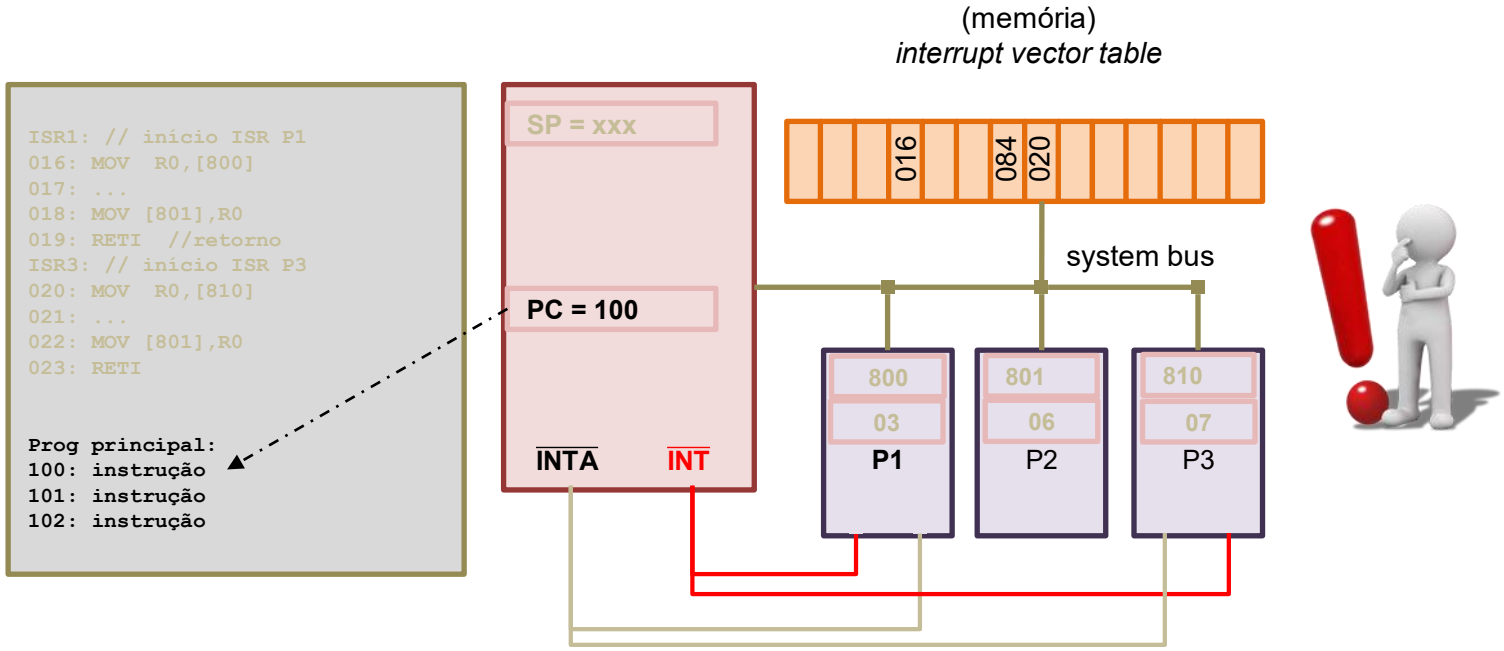




- Estudem o que vem para o futuro: The ARM **Advanced Microcontroller Bus Architecture (AMBA)**
- Conceito novo em DMA é Multilayer Bus Matrix – com vários canais de DMA, vários mestres e escravos!

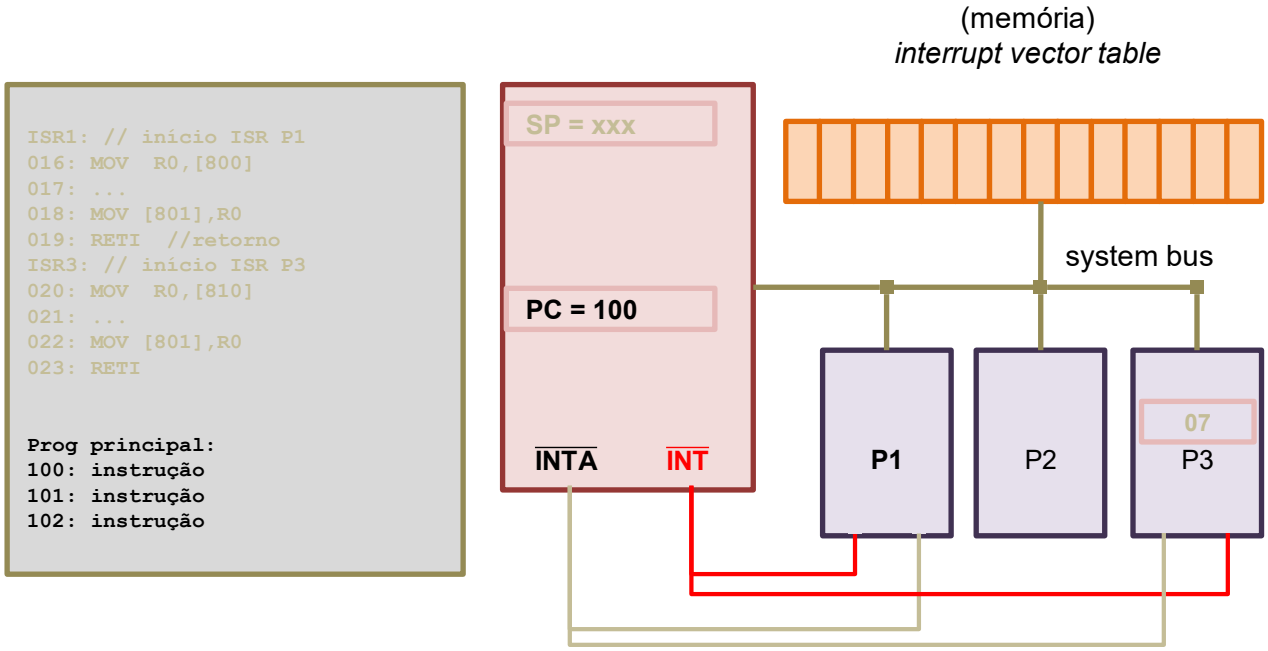






1.a) **μP** está executando a instrução 100
1.b) **P1** ativa a linha de pedido de interrupção (ou ativa DRQ1 ao controlador de DMA)
1.c) **P3** ativa a linha de pedido de interrupção (ou ativa DRQ3 ao controlador de DMA)

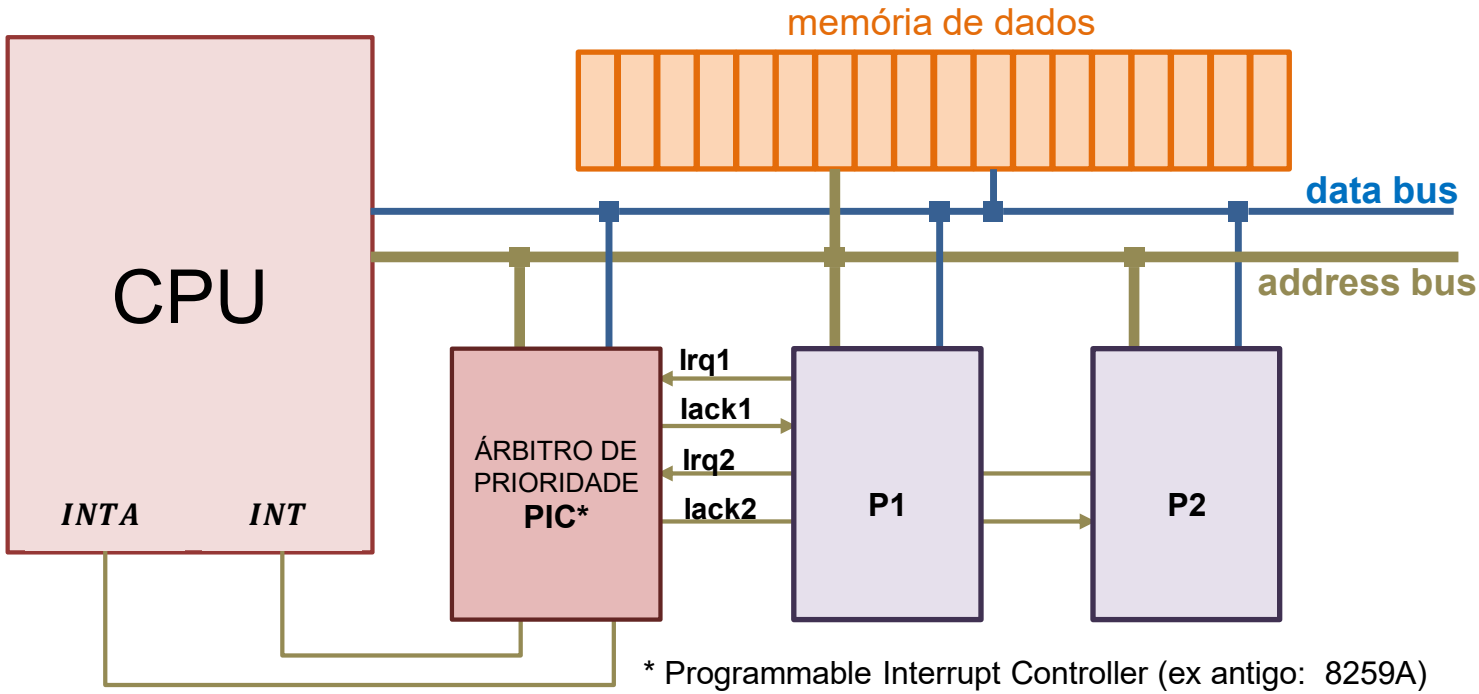
Qual das ISRs deve ser disparada? Para atender P1 ou P3?



- As situações de pedidos de ISR simultâneos requerem ***ARBITRAGEM***
- Vários métodos de arbitragem:
  - Arbitragem por prioridade (no caso acima, adicionamos um controlador de interrupções)
  - Arbitragem serial linear (daisy-chain)
- Métodos de arbitragem orientados para redes

➢ Como funciona (usando um exemplo de *interrupção vetorizada*):

a) A CPU está executando o programa principal (como anteriormente discutido)
b) Dois periféricos ativam cada qual seu pedido de IRQx para o controlador (árbitro) de interrupções (PIC)
c) O PIC ativa o pedido de interrupções INT– ao final da instrução a CPU atende ativando INTA (*interrupt acknoledge*)
d) O árbitro (PIC) envia IACKx para 1 periférico (o escolhido colocará seu vetor de interrupções no *data bus*)
e) A CPU recebe apenas o vetor do periférico escolhido e ativa a ISR que atende a este periférico
f) Ao final da ISR a CPU recupera os valores dos registradores e retorna ao programa principal...

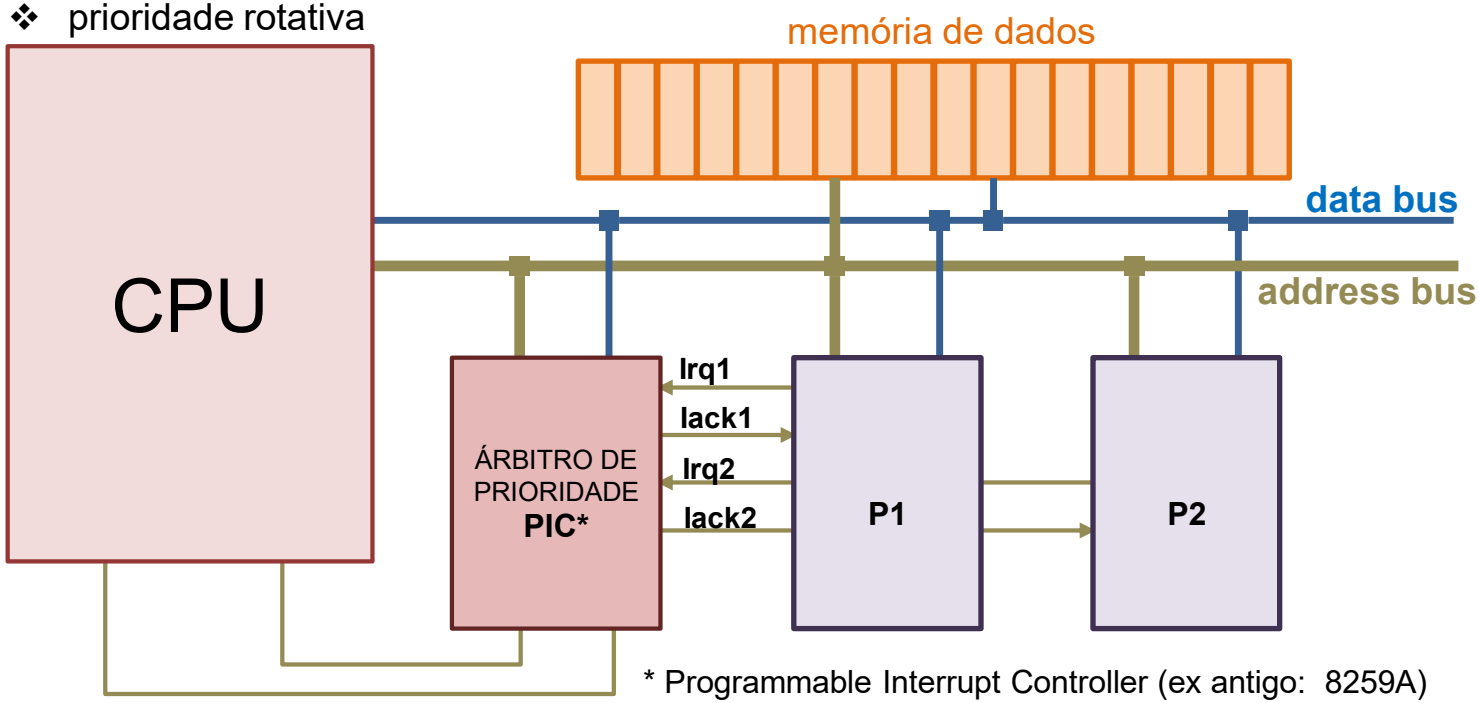


* Programmable Interrupt Controller (ex antigo: 8259A)

➢ Como funciona (usando um exemplo de *interrupção vetorizada*):

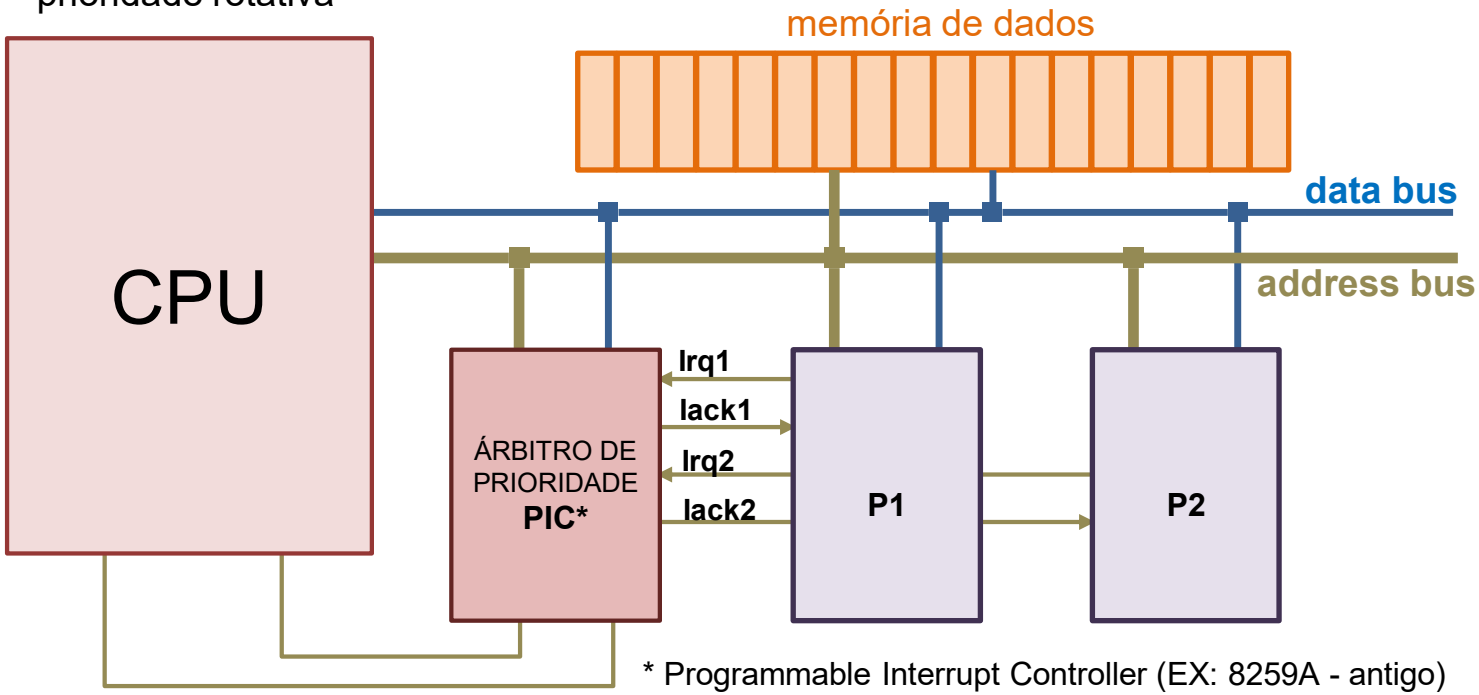
a) A CPU está executando o programa principal (como anteriormente discutido)
b) Dois periféricos ativam cada qual seu pedido de IRQx para o controlador (árbitro) de interrupções (PIC)
c) O PIC ativa o pedido de interrupções INT– ao final da instrução a CPU atende ativando INTA (*interrupt acknoledge*)
d) O árbitro (PIC) envia IACKx para 1 periférico (o escolhido colocará seu vetor de interrupções no *data bus*)
e) A CPU recebe apenas o vetor do periférico escolhido e ativa a ISR que atende a este periférico
f) Ao final da ISR a CPU recupera os valores dos registradores e retorna ao programa principal...

➢ Duas estratégias que o ÁRBITRO usa para escolher qual periférico será atendido:
- ❖ prioridade fixa
- ❖ prioridade rotativa



* Programmable Interrupt Controller (ex antigo: 8259A)

- Duas estratégias que o ÁRBITRO usa para escolher qual periférico será atendido:
  - prioridade fixa
  - prioridade rotativa

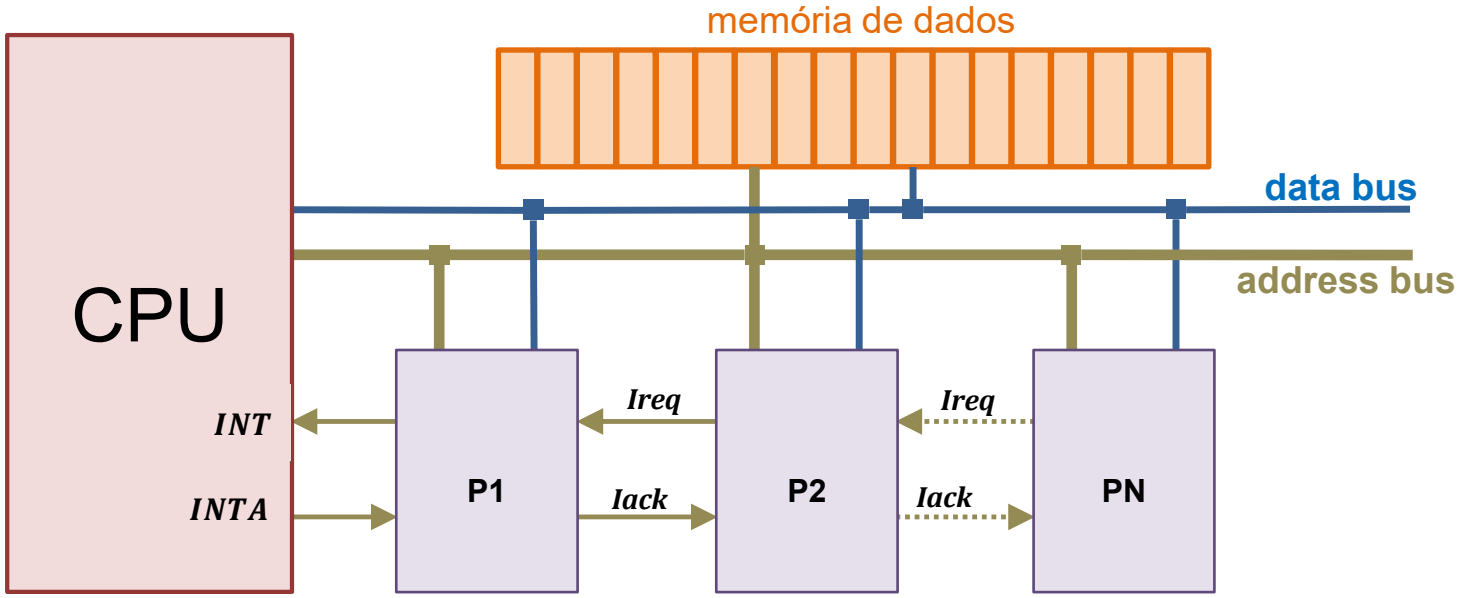


* Programmable Interrupt Controller (EX: 8259A - antigo)

Prioridade fixa: um ranking (configurado pela CPU) aponta quais periféricos tem maior prioridade. Estratégia preferida quando há clara diferença na prioridade por periférico. Contudo, periféricos com rank alto recebem muito mais serviços que os de baixo rank.

Prioridade rotativa (round-robin): nessa estratégia o árbitro muda a prioridade dos periféricos baseado no histórico de serviços prestados ao periférico. O último que recebeu atendimento terá menor prioridade. A estratégia rotativa pode parecer mais atraente, contudo, quando há muitos periféricos, pode passar um periférico de alta prioridade para o final da fila enquanto um de baixíssima prioridade pode receber serviço desnecessário.

- No método de arbitragem pela cadeia serial (daisy-chain) a arbitragem é construída no próprio periférico.

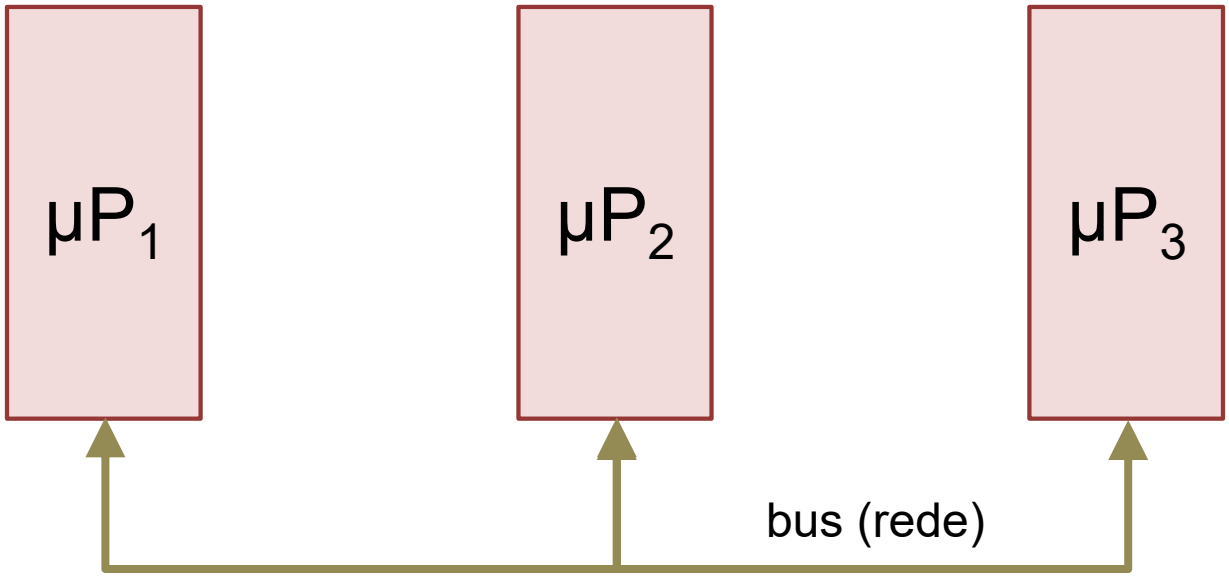


- Cada periférico tem ***Ireq*** e ***Iack*** (saída) e ***Ireq*** e ***Iack*** (entrada) – são conectados em cascata!

- Um periférico ativa ***Ireq*** se: a) quiser ele próprio um ISR, ou b) se sua ***entrada Ireq*** estiver ativa;
- O periférico mais perto da CPU tem seu ***Ireq*** ligado ao ***INT*** da CPU.
- Quando a CPU ativa ***INTA*** (*int acknowledge*) o periférico: a) coloca seu vetor no *data bus* (*se ele pediu ISR*) ou, b) propaga o sinal ***INTA*** na cadeia, passando para o periférico seguinte.

- claro, o periférico na frente da cadeia (mais perto da CPU) terá prioridade porque não propagará ***INTA*** aos demais periféricos caso ele tenham solicitado ISR ao mesmo tempo que outro periférico.

Pros: podemos adicionar mais periféricos sem redesenhar o sistema (PIC tem um número fixo de canais)
Cons: não suporta esquemas avançados de prioridade, e se algum periférico falhar o restante da cadeia pode perder funcionalidade.



- Às vezes múltiplos processadores podem estar conectados no mesmo barramento, formando redes.
- A arbitragem desse tipo de rede é tipicamente embutida no protocolo da rede (como veremos).
- Ex: Um processador em Ethernet ou em I2C pode enviar uma série de bits sem detectar que outro processador está também tentando transmitir por aquele canal. Nesses casos em que ocorre colisão de dados os processadores param de transmitir e esperam por algum tempo aleatório...
- Protocolos de redes serão vistos na aula seguinte...

➢ **Desvantagens** de uso de ***barramento*** (comunicação paralela)***:***

✓ *Gargalo*: a largura de banda (***bandwidth***) do barramento limita o máximo de informação (***throughput***) que flui para os dispositivos I/O;

✓ *Velocidade máxima* do barramento é limitada por:

→ comprimento do barramento (*longos fios paralelos = capacitâncias maiores, os sinais começam a levar mais tempo para subir ou descer*) – regra: *curtas distâncias*;

→ desalinhamento entre sinais (*longos barramentos aumentam as variações das subidas e descidas dos sinais, isso causa desalinhamento de sinais que deveriam chegar simultaneamente ao destino*)

→ o número de dispositivos no barramento (*mais dispositivos mais distorções nos sinais*);

→ suporte a ampla gama de dispositivos com varias latências e taxas de transferências de dados diferentes;

→ custo mais alto;

→ mais volumosos (*dado que são necessários vários fios em paralelo*).

➢ **Vantagens** de uso de ***barramento*** (comunicação paralela):

✓ *Versasitilidade*: novos dispositivos podem ser inseridos facilmente;

✓ *Baixo custo*: um conjunto de fios é compartilhado por vários dispositivos de múltiplas formas;

✓ Multiplos fios (endereço, controle, data) enviam *1 bit por fio* (*fluência alta – high throughput*);

✓ Geralmente usados para conectar dispositivos no mesmo CI ou na mesma PCB

Obrigado...

Até a próxima aula.