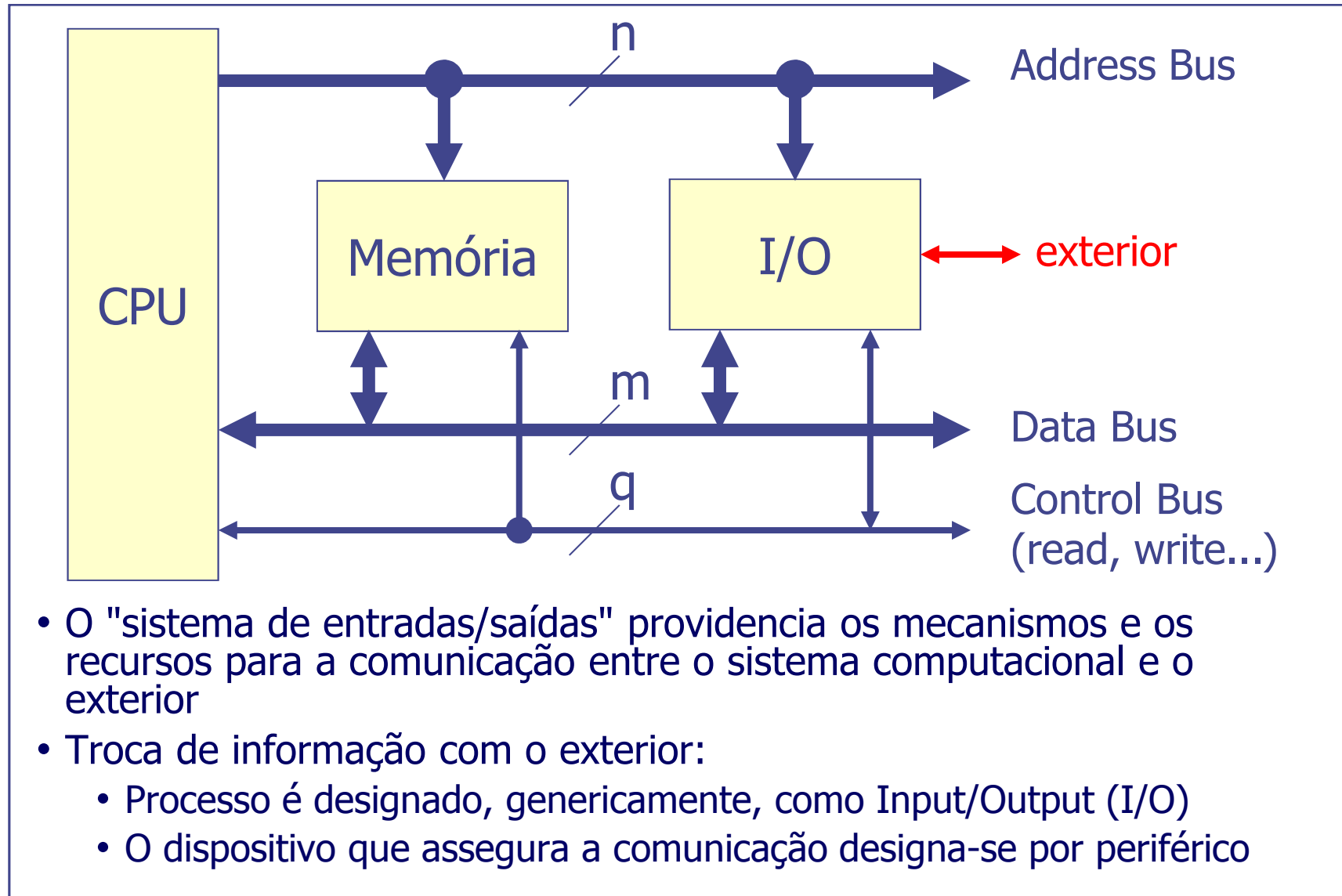


## Aulas 3, 4 e 5

- Noção de periférico; estrutura básica de um módulo de I/O; modelo de programação
- Endereçamento das unidades de I/O
- Descodificação de endereços e geração de sinais de seleção de memória e unidades de I/O
- Mapeamento no espaço de endereçamento de memória
- Exemplo de um gerador de sinais de seleção programável
- Estrutura básica de um porto de I/O de 1 bit no microcontrolador PIC32. Estrutura dos portos de I/O de "n" bits.

José Luís Azevedo, Arnaldo Oliveira, Tomás Silva, Bernardo Cunha

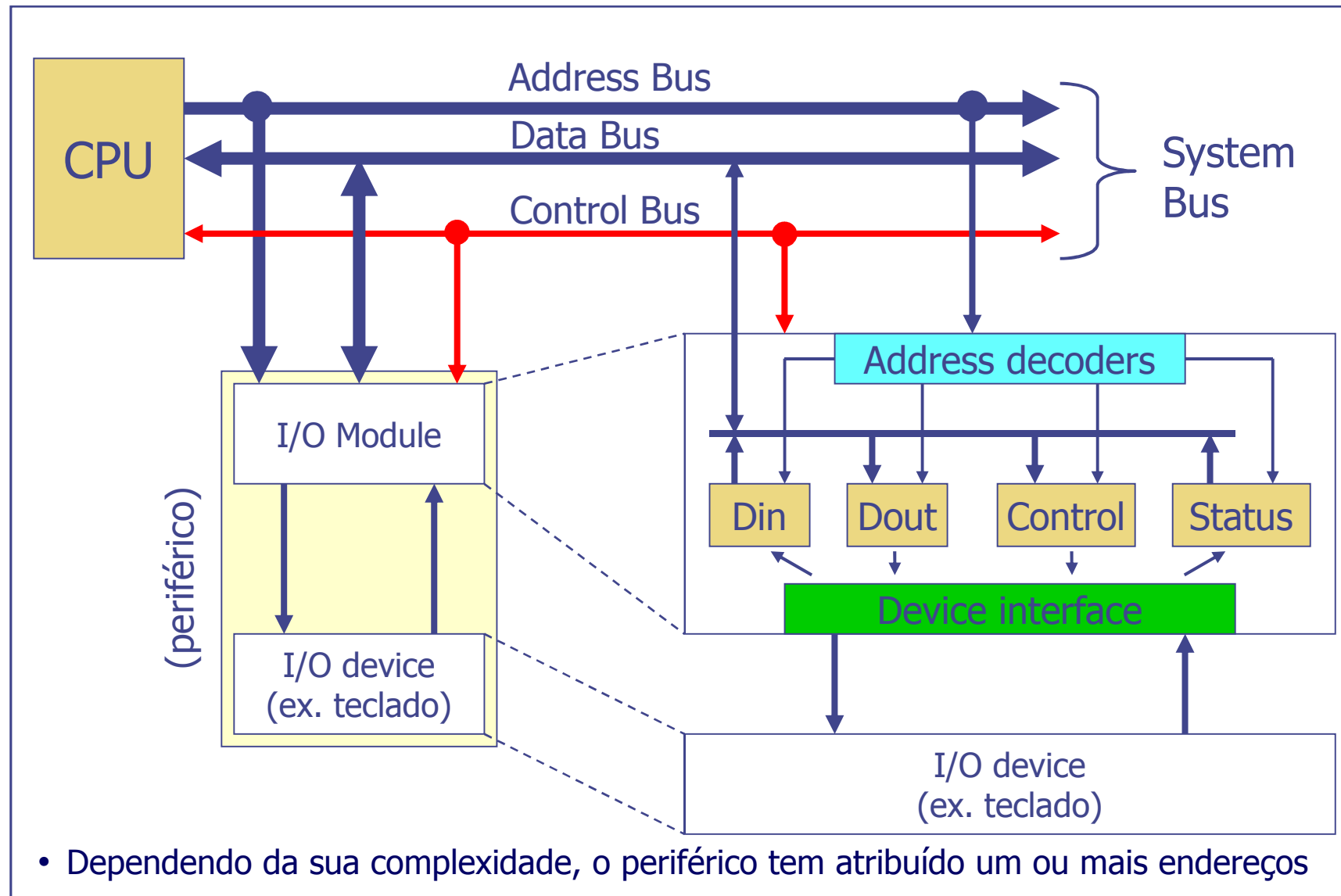
# Introdução



# Introdução

- Dispositivos periféricos:
  - grande variedade (por exemplo: teclado, rato, unidade de disco ...)
  - com métodos de operação diversos
  - assíncronos relativamente ao CPU
  - geram diferentes quantidades de informação com diferentes formatos a diferentes velocidades (de alguns bits/s a dezenas de Megabyte/s)
  - mais lentos que o CPU e a memória
- É necessária uma interface que providencie a adaptação entre as características intrínsecas do dispositivo periférico e as do CPU / memória
- **Módulo de I/O**

# Módulo de I/O



# Módulo de I/O

- O módulo de I/O pode assim ser visto como um módulo de compatibilização entre as características e modo de funcionamento do sistema computacional e o dispositivo físico propriamente dito.
- Ao nível do hardware:
  - Adequa as características do dispositivo físico de I/O às características do sistema digital ao qual tem que se ligar. O periférico liga-se ao sistema através dos barramentos, do mesmo modo que todos os outros dispositivos (ex. memória)
- Interação com o dispositivo físico:
  - Lida com as particularidades do dispositivo, nomeadamente, formatação de dados, deteção e gestão de situações de erro, ...
- Ao nível do software:
  - Adequa o dispositivo físico à forma de organização do sistema computacional, disponibilizando e recebendo informação através de registos; esta solução esconde do processador a complexidade e os detalhes de implementação do dispositivo periférico

# Módulo de I/O

- O módulo de I/O permite ao processador ver um modelo simplificado do periférico, escondendo os detalhes de funcionamento interno
- Com a adoção do módulo de I/O, o dispositivo externo, independentemente da sua natureza e função, passa a ser encarado pelo processador como uma coleção de registos de dados, de controlo e de *status*
- A comunicação entre o processador e o periférico é assegurada por operações de escrita e de leitura, em tudo semelhantes a um acesso a uma posição de memória
  - Ao contrário do que acontece na memória, o valor associado a estes endereços pode mudar sem intervenção do CPU
- O conjunto de registos e a descrição de cada um deles são específicos para cada periférico e constituem o que se designa por **modelo de programação do periférico**

# Módulo de I/O – modelo de programação

- **Data Register(s)** (*Read/Write*)
  - Registo(s) onde o processador coloca a informação a ser enviada para o periférico (*write*) e de onde lê informação proveniente do periférico (*read*)
- **Status Register(s)** (*Read only*)
  - Registo(s) que engloba(m) um conjunto de bits que dão informação sobre o estado do periférico (ex. operação terminada, informação disponível, situação de erro, ...)
- **Control Register(s)** (*Write only* ou *Read/Write*)
  - Registo(s) onde o CPU escreve informação sobre o modo de operação do periférico (comandos)
- É comum um só registo incluir as funções de controlo e de *status*. Nesse caso, um conjunto de bits desse registo está associado a funções de controlo (*read/write* ou *write only bits*) e outro conjunto a funções de status (*read only bits*)

# Comunicação entre o CPU e outros dispositivos

- A iniciativa da comunicação é sempre do CPU, no contexto da execução das instruções
- A comunicação entre o CPU e um dispositivo externo genérico envolve a existência de um **protocolo** que ambas as partes conhecem e respeitam
- Definem-se, assim, duas operações básicas:
  - **Write** (fluxo de informação: CPU → dispositivo externo)
  - **Read** (fluxo de informação: CPU ← dispositivo externo)
- Uma operação de acesso do CPU a um dispositivo externo envolve:
  - Usar o barramento de endereços para especificar o **endereço do dispositivo a aceder**
  - Usar o barramento de controlo para sinalizar qual a operação a realizar (*read* ou *write*)
  - O barramento de dados assegura a transferência de dados, no sentido adequado, entre as duas entidades envolvidas na comunicação

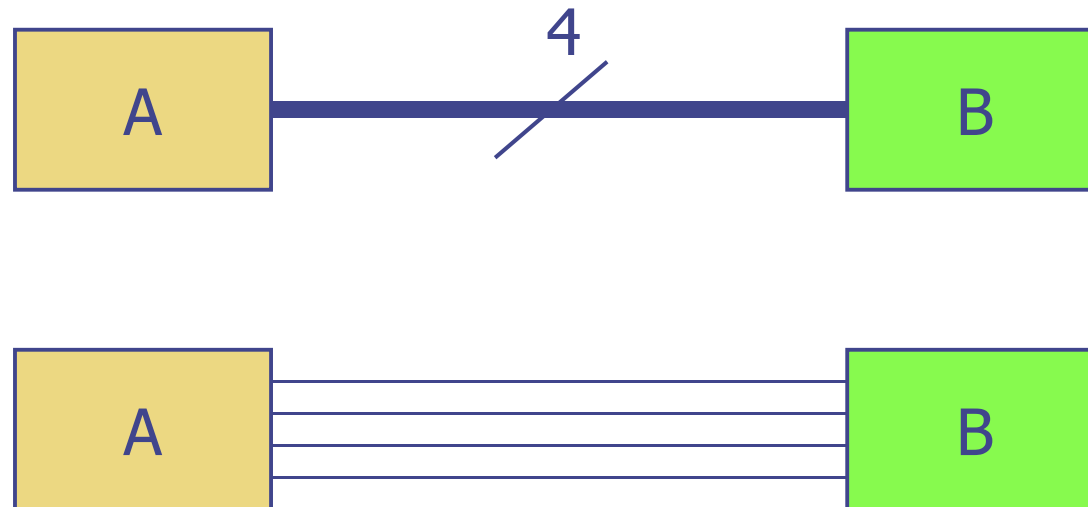


# Seleção do dispositivo externo

- **Operação de escrita** (CPU → dispositivo externo)
  - apenas 1 dispositivo deve receber a informação colocada pelo CPU no barramento de dados
- **Operação de leitura** (CPU ← dispositivo externo)
  - apenas 1 dispositivo pode estar ativo no barramento de dados
  - os dispositivos, quando inativos, têm que estar eletricamente desligados do barramento
  - é obrigatório utilizar circuitos com saída **Tri-State**
- Num sistema computacional há múltiplos circuitos ligados ao barramento de dados
  - unidades de I/O
  - circuitos de memória
- Há, pois, necessidade de, a partir do endereço gerado pelo CPU, **selecionar apenas um** dos vários dispositivos existentes no sistema

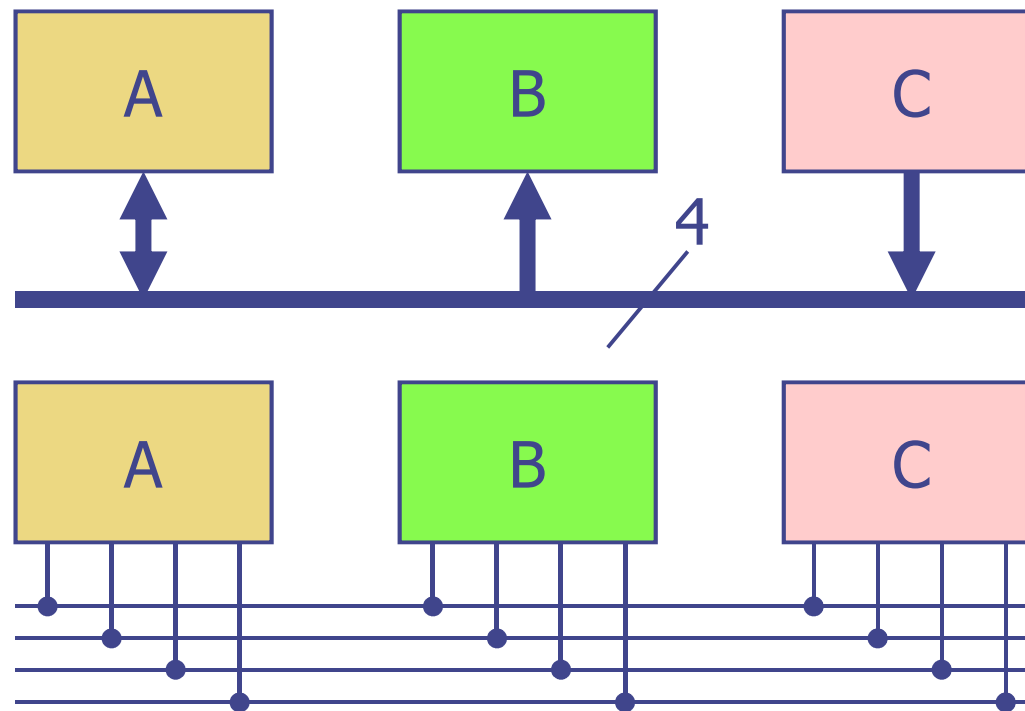
# Barramento simples (revisão)

- Barramento (bus) - conjunto de ligações (fios) agrupadas, geralmente, segundo uma dada função; cada ligação transporta informação relativa a 1 bit.
- Exemplo – barramento de 4 bits que liga os dispositivos A e B



# Barramento partilhado (revisão)

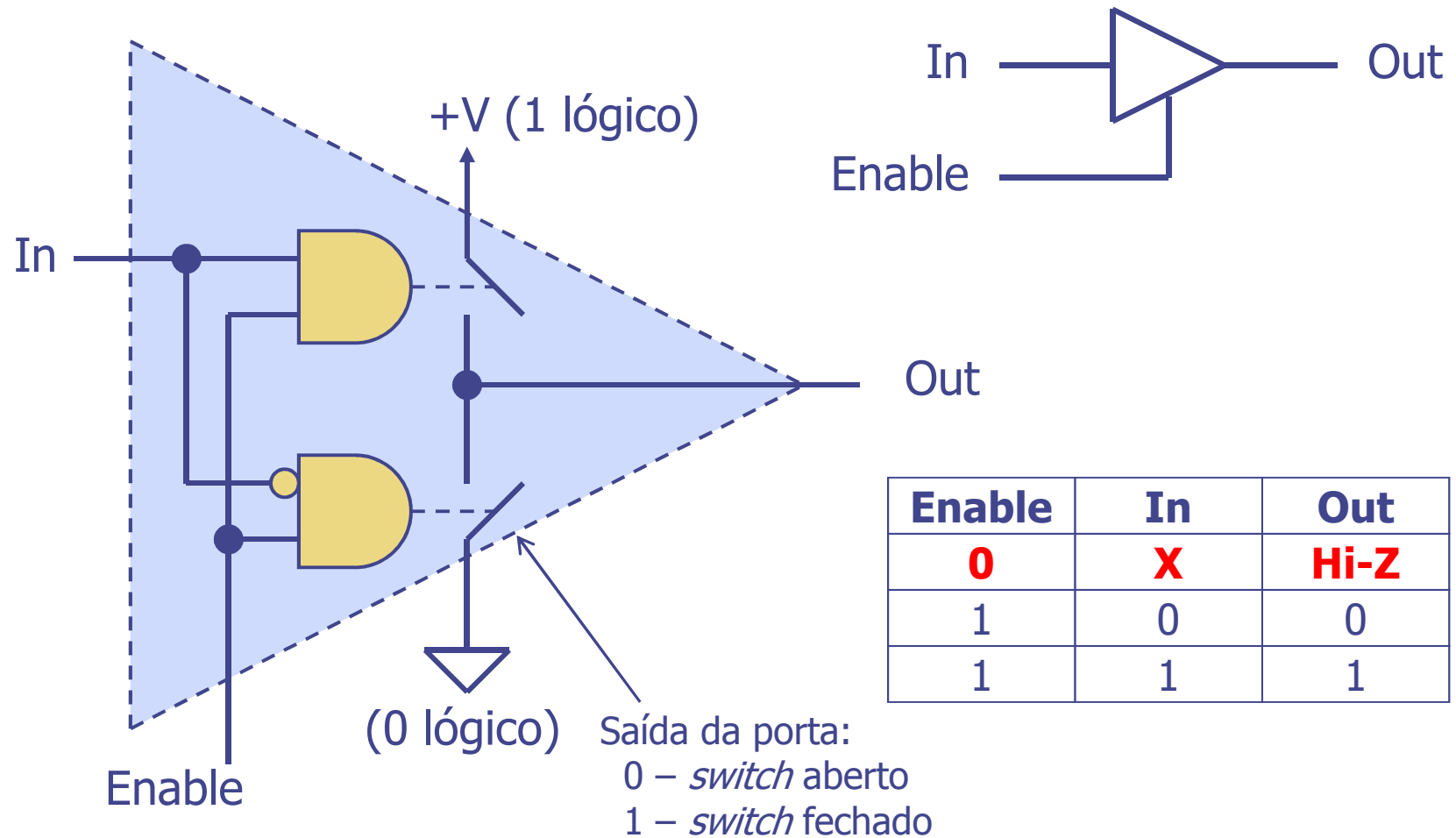
- Barramento partilhado (*shared bus*) - barramento que interliga vários blocos do sistema de computação
- Exemplo: barramento de interligação entre os blocos funcionais A, B e C.



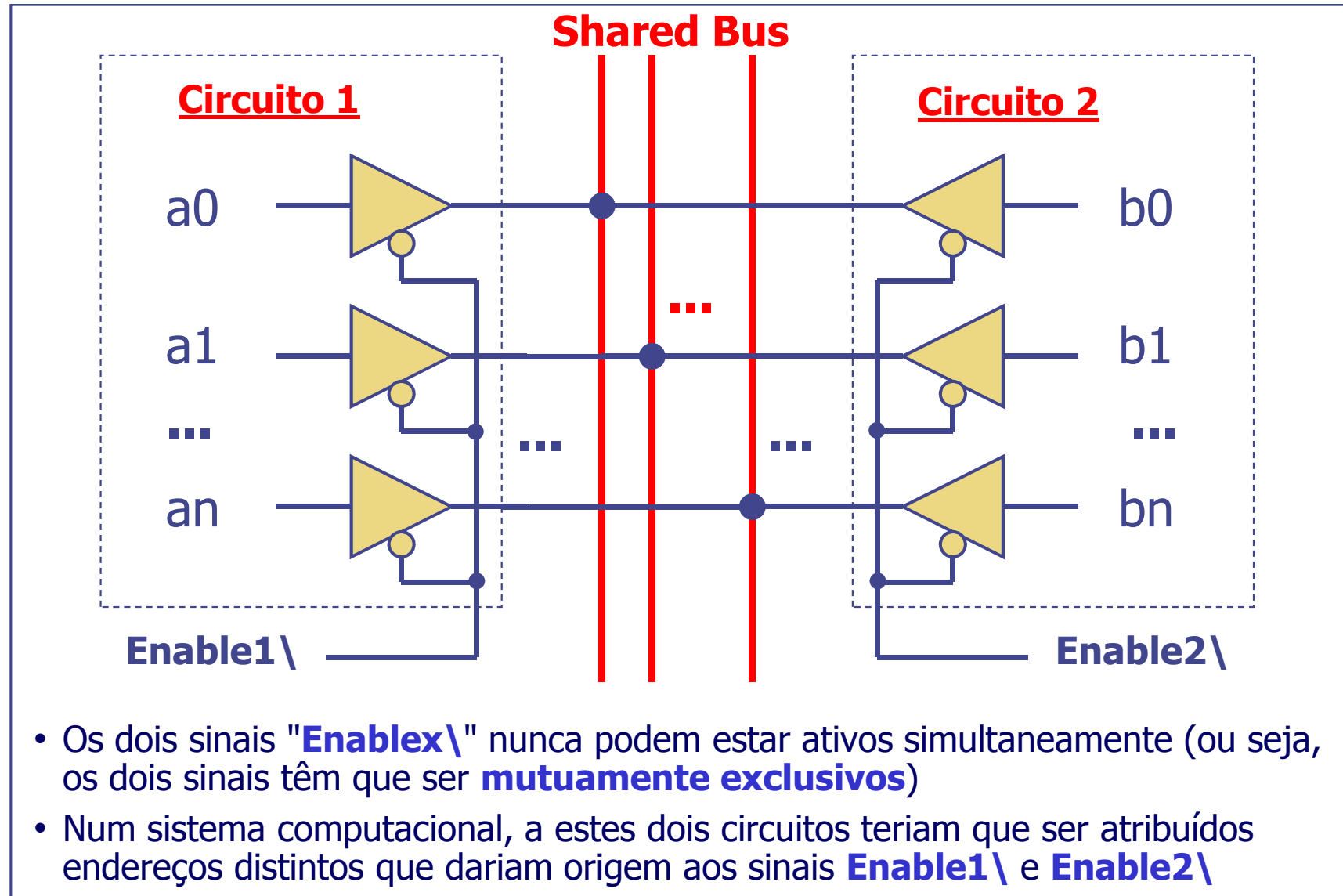
- Neste exemplo, a comunicação pode realizar-se de A para B, de C para A ou de C para B

# Porta *Tri-State*

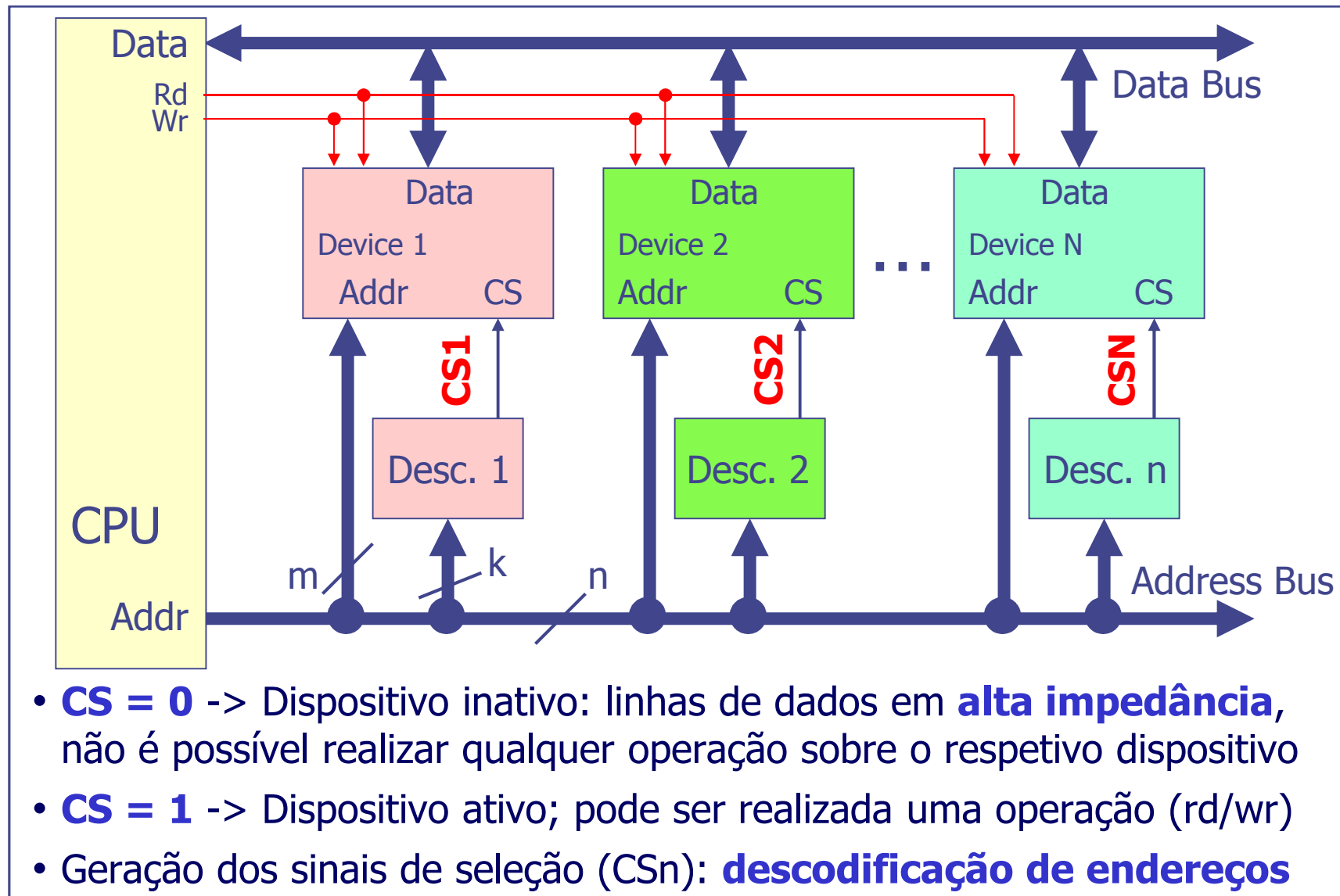
- Modelo de funcionamento de uma porta *Tri-State*



# Ligação a um barramento partilhado



# Seleção do dispositivo externo



# Seleção do dispositivo externo

- Para ser possível o acesso individualizado a todos os recursos disponíveis, o dispositivo externo pode necessitar de apenas um endereço ou de uma gama contígua de endereços.
- Exemplos:
  - Para aceder a um porto de saída de 1 byte (por exemplo implementado como um registo de 8 bits) será apenas necessário 1 endereço
  - Para aceder a todas as posições de uma memória de 1kB são necessários, numa organização *byte-addressable*, 1024 endereços consecutivos (10 bits do barramento de endereços)
  - Para aceder a todos os recursos de um periférico com 5 registos internos, de 1 byte cada, são necessários 5 endereços distintos (3 bits do barramento de endereços)
- A implementação do decodificador de endereços para um dado dispositivo tem que ter em consideração, entre outros aspetos, as necessidades de endereçamento desse dispositivo

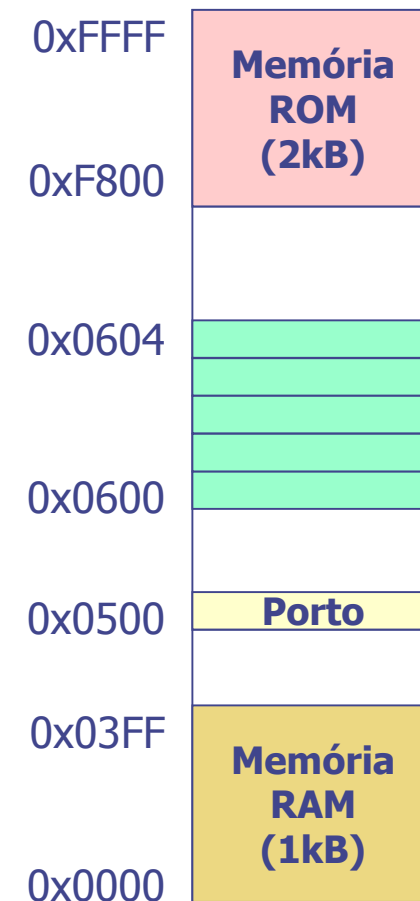
# Mapeamento no espaço de endereçamento

- Exemplo de mapeamento de dispositivos no espaço de endereçamento do microprocessador:

- Espaço de endereçamento de 16 bits ( $2^{16} = 64k$ ,  $A_{15}-A_0$ )
- Organização *byte-addressable*
- Dispositivos a mapear:
  - porto de saída de 8 bits,
  - memória RAM de 1k x 8 (1 kB) – 10 bits
  - memória ROM de 2k x 8 (2 kB) – 11 bits
  - periférico com 5 registos de 8 bits – 3 bits

- Com estes dados, uma possível solução para o mapeamento dos 4 dispositivos é:

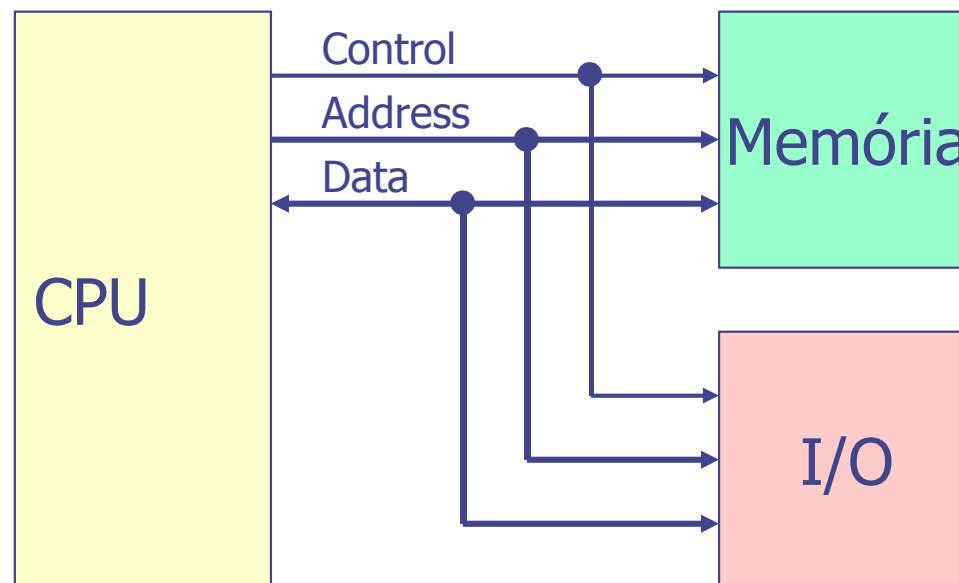
- Memória de 1kB: [0x0000, 0x03FF]
- Porto de 1 byte: 0x0500
- Periférico: [0x0600, 0x0604]
- Memória de 2kB: [0xF800, 0xFFFF]





# Endereçamento das unidades de I/O

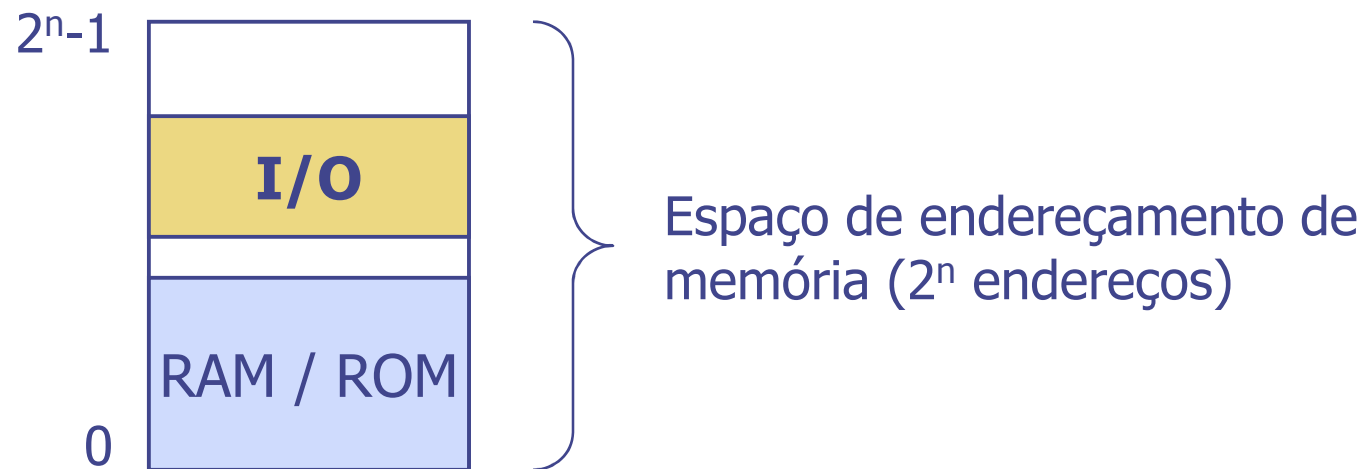
- *Memory-mapped I/O*



- Memória e unidades de I/O coabitam no mesmo espaço de endereçamento
- Uma parte do espaço de endereçamento é reservada para periféricos

# Endereçamento das unidades de I/O

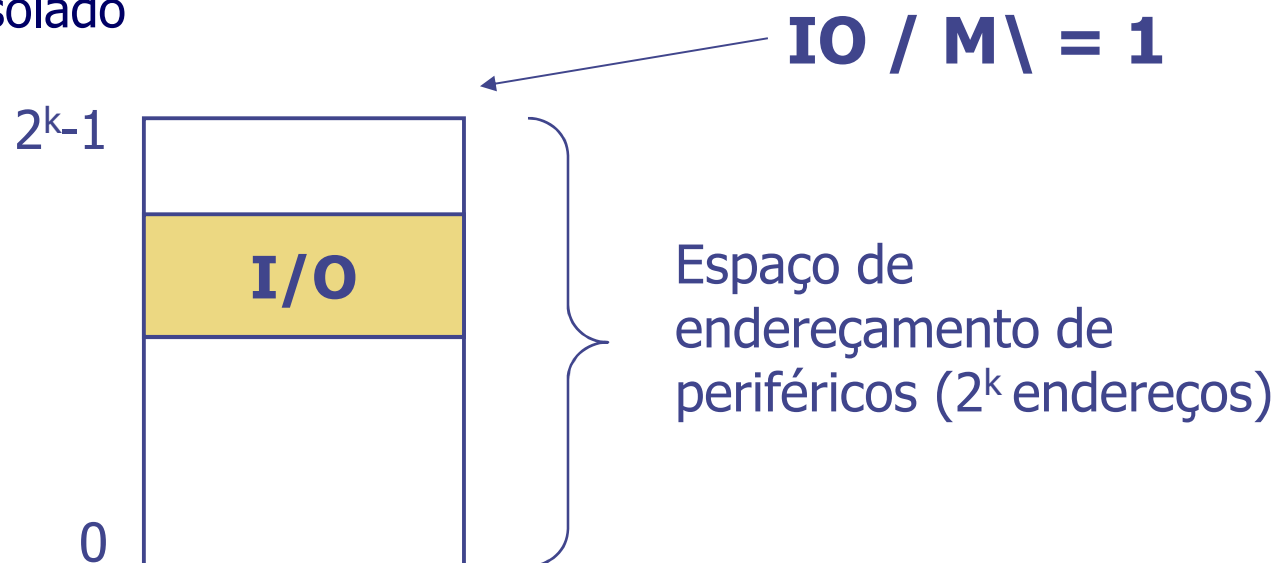
- *Memory-mapped I/O*



- Às unidades de I/O são atribuídos endereços do espaço de endereçamento de memória
- O acesso às unidades de I/O é feito com as mesmas instruções com que se acede à memória (**lw**, **lb**, **sw** e **sb** no caso do MIPS)

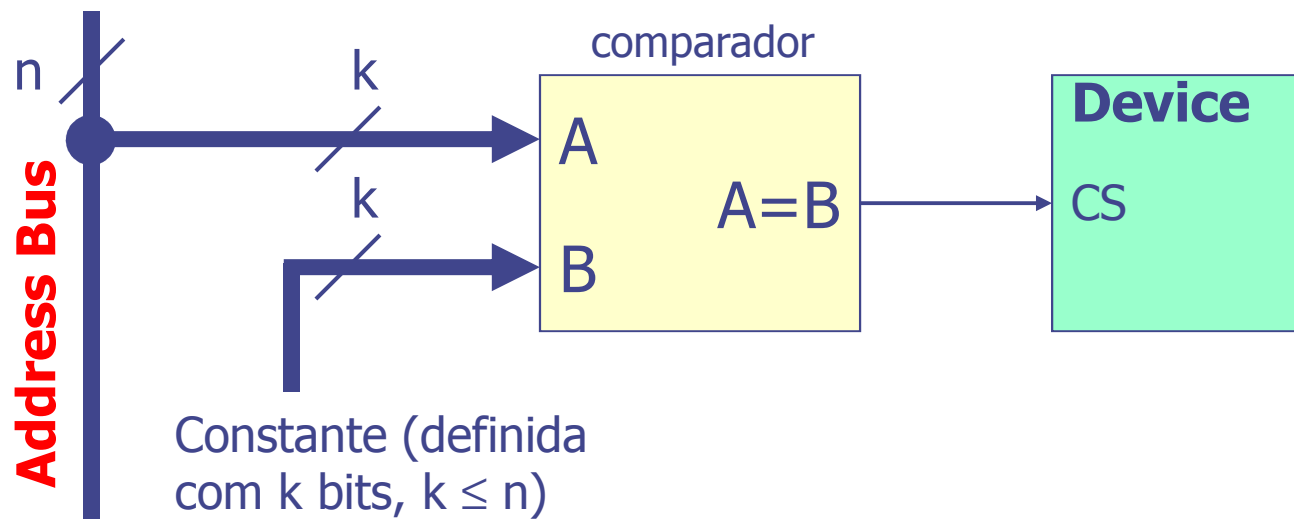
# Endereçamento das unidades de I/O

- I/O Isolado



- Memória e periféricos em espaços de endereçamento separados
- Sinal do barramento de controlo indica a qual dos espaços de endereçamento (I/O ou memória) se destina o acesso; por exemplo  $IO/M\$ :
  - $IO/M\=1$  -> acesso ao espaço de endereçamento de I/O
  - $IO/M\=0$  -> acesso ao espaço de endereçamento de memória
- O acesso às unidades de I/O é feito com instruções específicas

# Descodificação de endereços



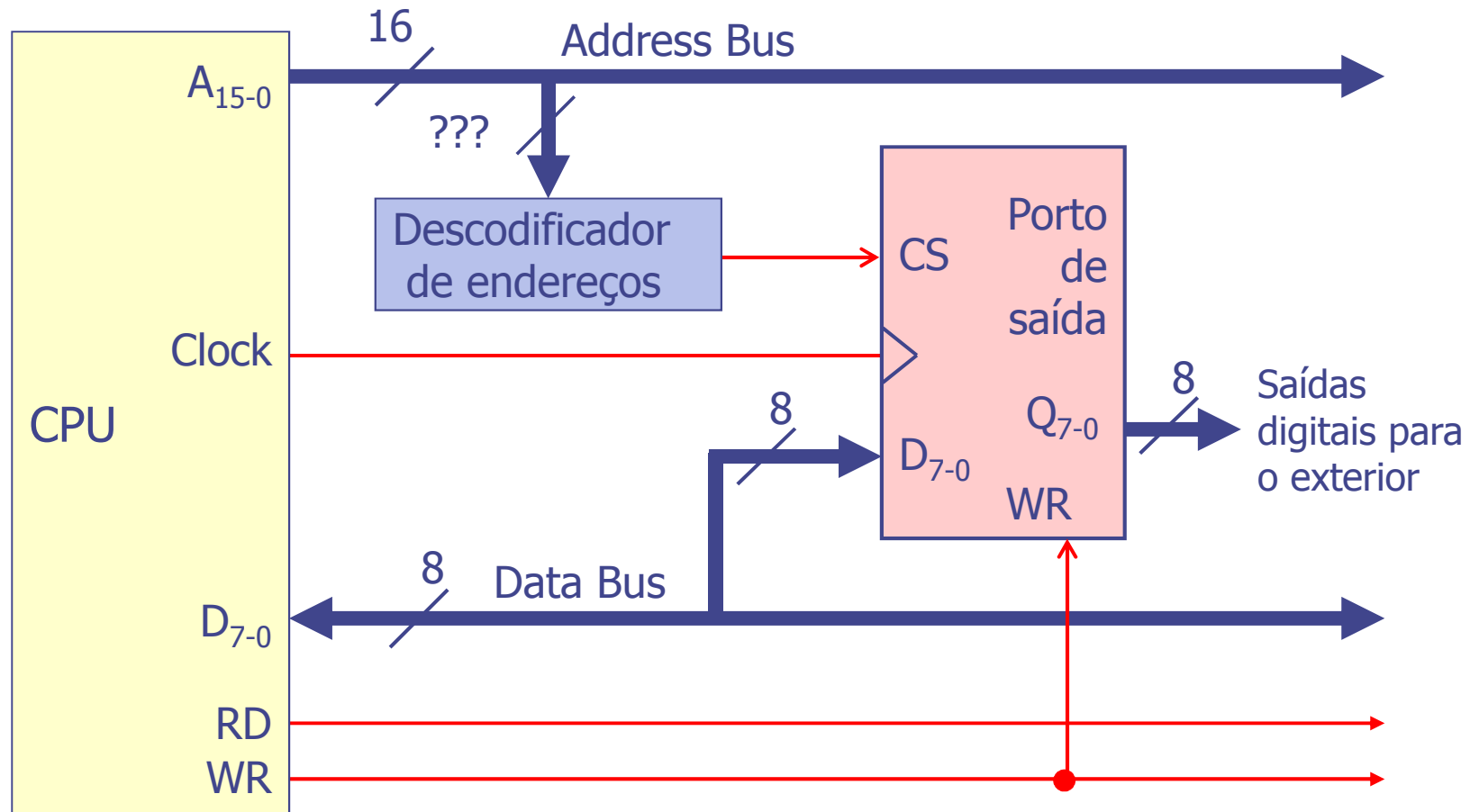
- O dispositivo é selecionado quando a combinação binária presente nos  $k$  bits do *address bus* for igual à constante (entrada B)
- "CS": *Chip Select* (ou "CE" – *Chip Enable*)

# Descodificação de endereços

- Supondo um CPU com um barramento de endereços de 16 bits e um barramento de dados de 8 bits
  - 16 bits ( $A_{15}-A_0$ )  $\rightarrow (2^{16}=64 \text{ k})$
  - 8 bits ( $D_7-D_0$ )
- **Exemplo 1:** ligação de um porto de saída de 1 byte ao CPU
- **Exemplo 2:** ligação de uma memória de 1 kByte ao CPU
  - 1 kByte ( $1\text{k} \times 8$ ) - 10 bits de endereço ( $2^{10}=1\text{k}$ )

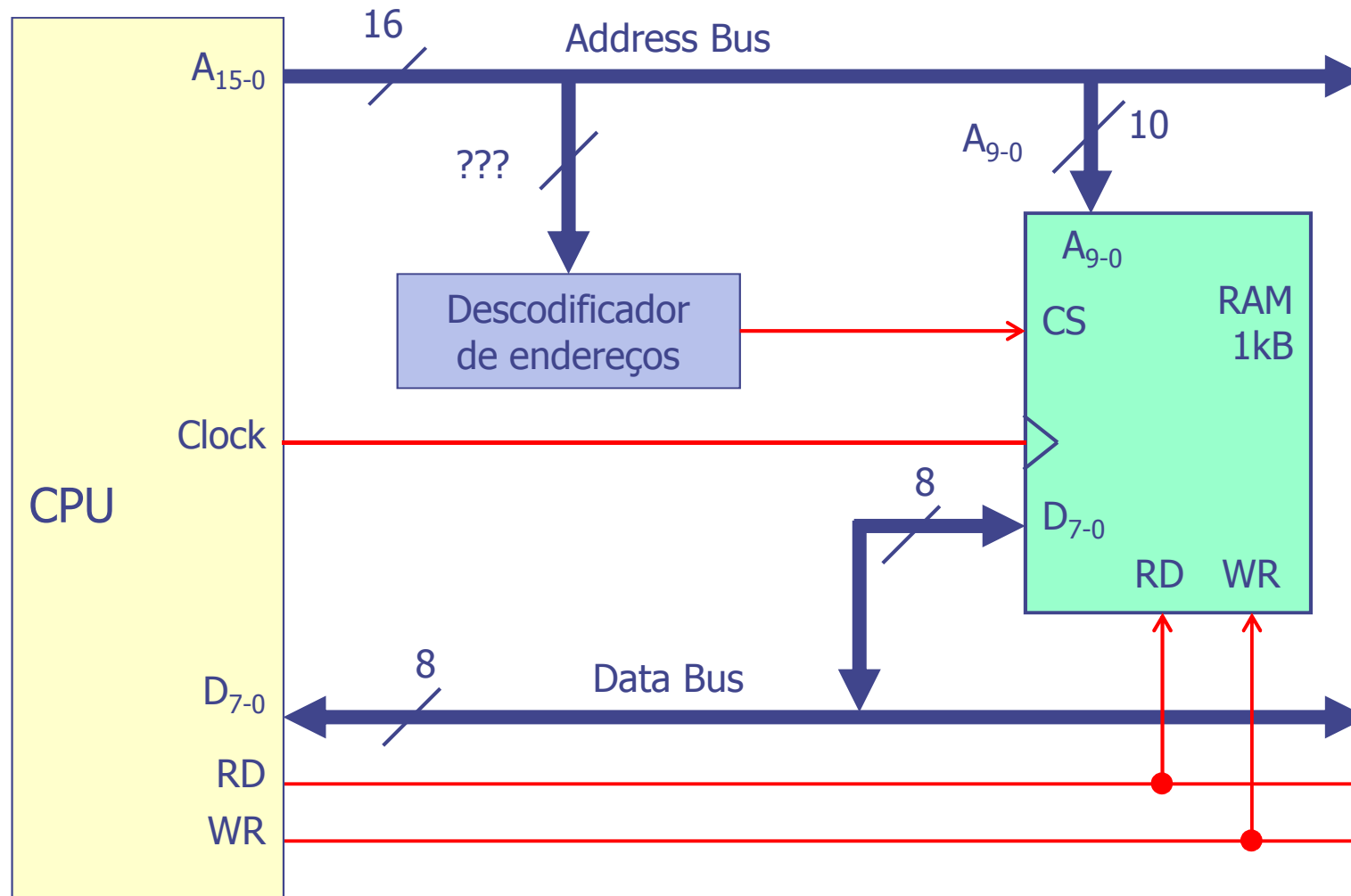
# Descodificação de endereços

- Exemplo 1: ligação de um porto de saída de 1 byte ao CPU



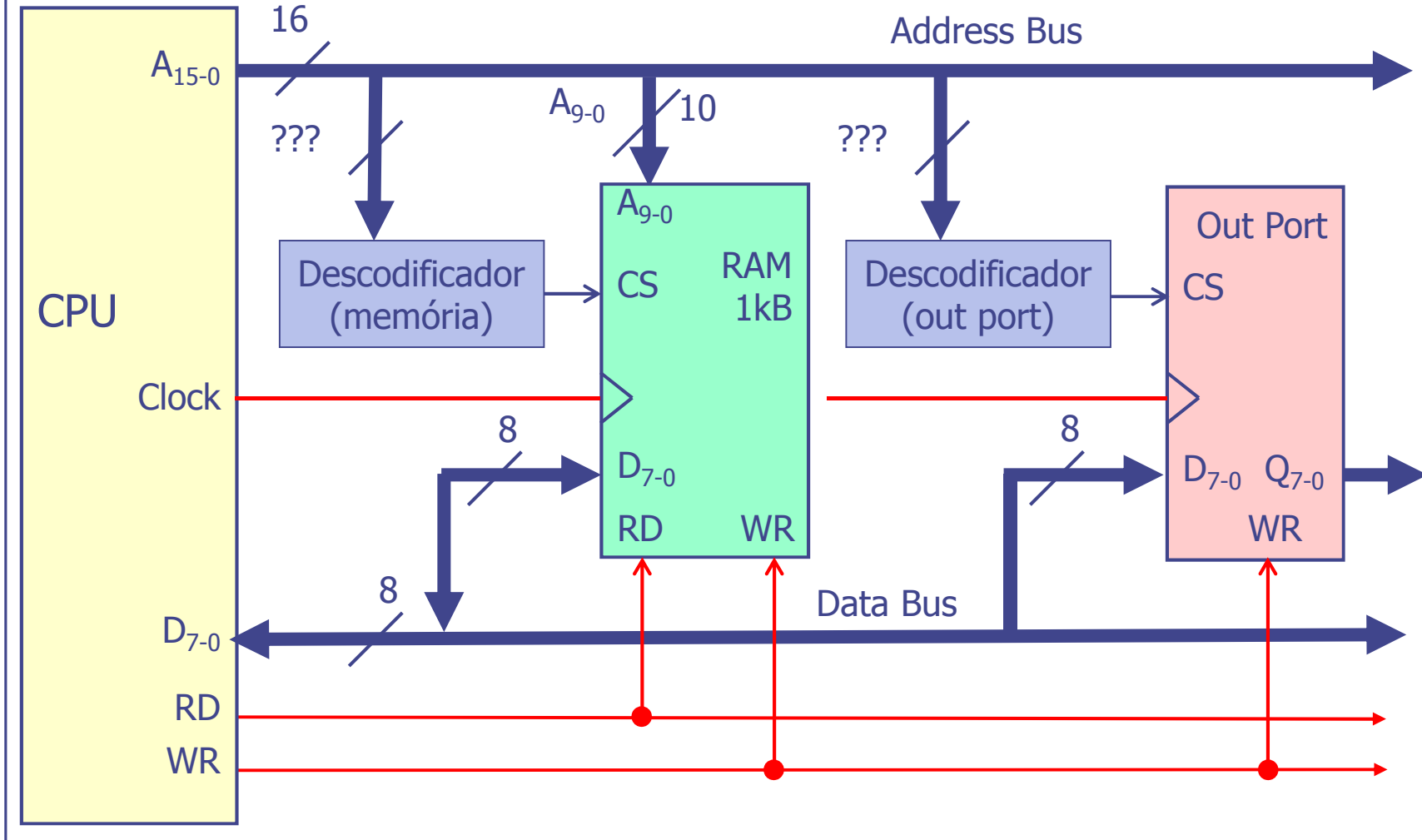
# Descodificação de endereços

- Exemplo 2: ligação de uma memória de 1 kByte ao CPU



# Descodificação de endereços

- Ligação do porto de saída e da memória





# Descodificação de endereços

- Descodificação total
  - Para uma dada posição de memória/periférico existe apenas um endereço possível para acesso
  - Todos os bits relevantes são descodificados
- Descodificação parcial
  - Vários endereços possíveis para aceder à mesma posição de memória/periférico
  - Apenas alguns bits são descodificados
  - Conduz a circuitos de descodificação mais simples (e menores atrasos)

# Descodificação de endereços

- Descodificador de endereços da memória do exemplo anterior (1k)
  - Supondo que se pretende a memória mapeada a partir do endereço 0x0000 do espaço de endereçamento de 16 bits, quantos bits será necessário descodificar? Quais são esses bits?
- Descodificador de endereços do porto de saída
  - Supondo que se pretende o porto mapeado no endereço 0x0500 do espaço de endereçamento de 16 bits, quantos bits será necessário descodificar? Quais são esses bits?

- Descodificação total para o caso do porto:

$$A0 \setminus \Leftrightarrow \overline{A0}$$

$$0x0500 = 0000 \text{ } 0\textcolor{red}{1}0\textcolor{red}{1} \text{ } 0000 \text{ } 0000$$

$$CS = A15 \setminus . A14 \setminus . A13 \setminus . A12 \setminus . A11 \setminus . \textcolor{red}{A10} . A9 \setminus . \textcolor{red}{A8} .$$

$$A7 \setminus . A6 \setminus . A5 \setminus . A4 \setminus . A3 \setminus . A2 \setminus . A1 \setminus . A0 \setminus$$

Qual a consequência de se remover da expressão acima o termo  $A0 \setminus$  ?

# Descodificação de endereços

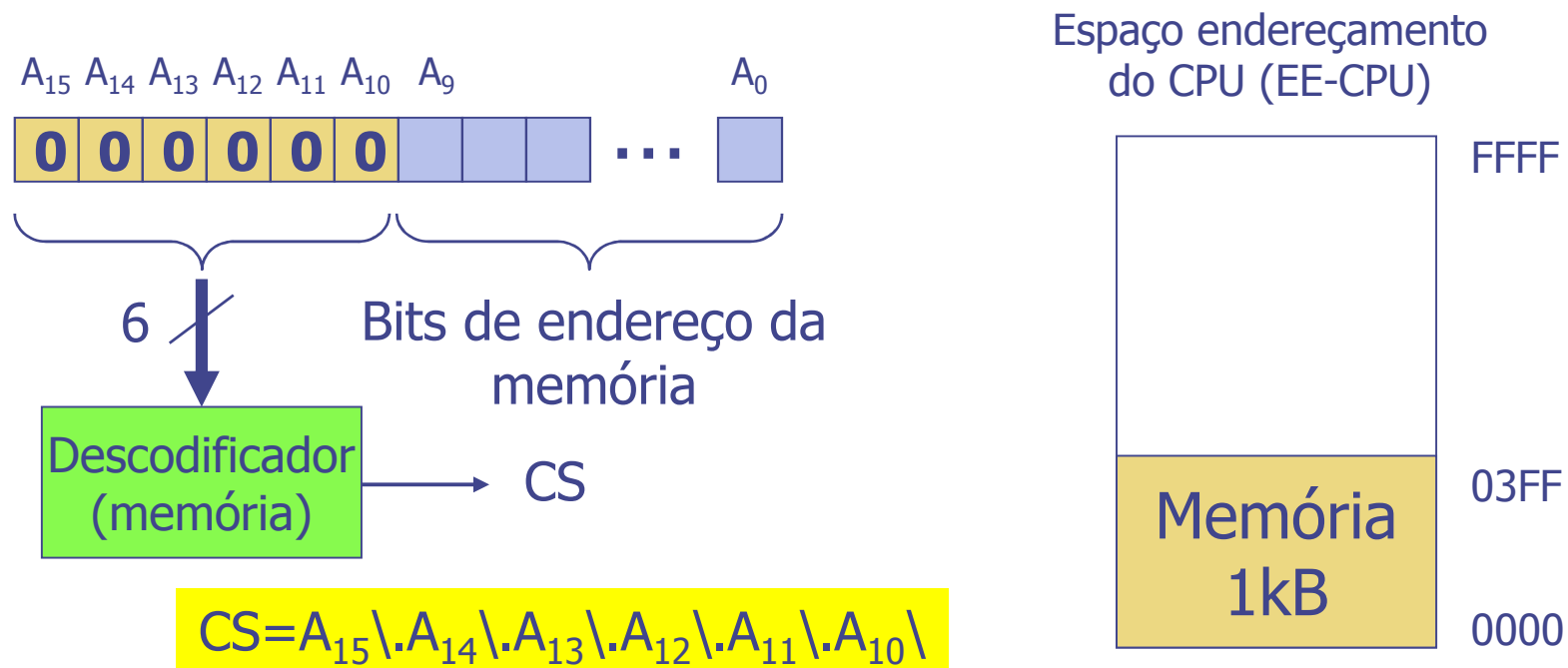
- Para o decodificador da memória do exemplo anterior



- Para garantir que a memória de 1kB está mapeada a partir do endereço 0x0000 (na gama 0x0000-0x03FF), há várias soluções possíveis. Vamos analisar as seguintes 3:
  - 1) decodificação total** – usar os 6 bits  $A_{15}$  a  $A_{10}$  (ex. **000000**)
  - 2) decodificação parcial** – usar apenas  $A_{13}$ ,  $A_{12}$ ,  $A_{11}$  e  $A_{10}$  e ignorar  $A_{15}$  e  $A_{14}$  (e.g. **xx0000**)
  - 3) decodificação parcial** – usar  $A_{15}$ ,  $A_{14}$ ,  $A_{13}$  e  $A_{12}$  e ignorar  $A_{11}$  e  $A_{10}$  (e.g. **0000xx**)
- Que implicações têm estas escolhas?

## Descodificação de endereços – descodificação total

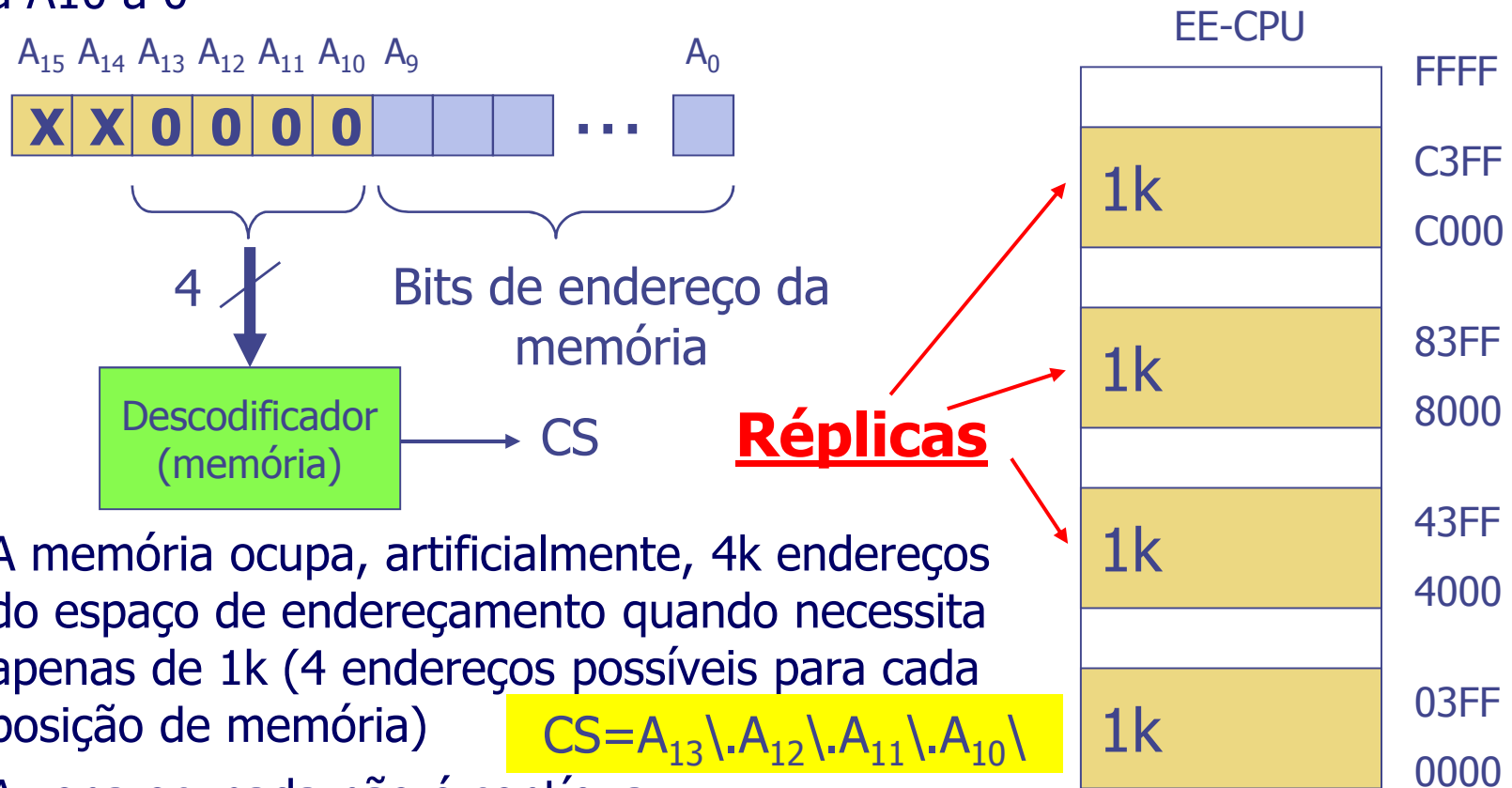
- Solução 1 – utilizar todos os bits possíveis, ex. **000000**. Isto significa que um endereço só é válido para aceder à memória se tiver os 6 bits mais significativos a 0



- A memória ocupa 1k do espaço de endereçamento
- Apenas 1 endereço possível para cada posição de memória

# Descodificação de endereços – descodificação parcial

- Solução 2 – usar A13, A12, A11 e A10 e ignorar A15 e A14, e.g. **xx0000**. Isto significa que um endereço válido para aceder à memória não depende do valor dos bits A15 e A14, mas tem que ter os bits A13 a A10 a 0

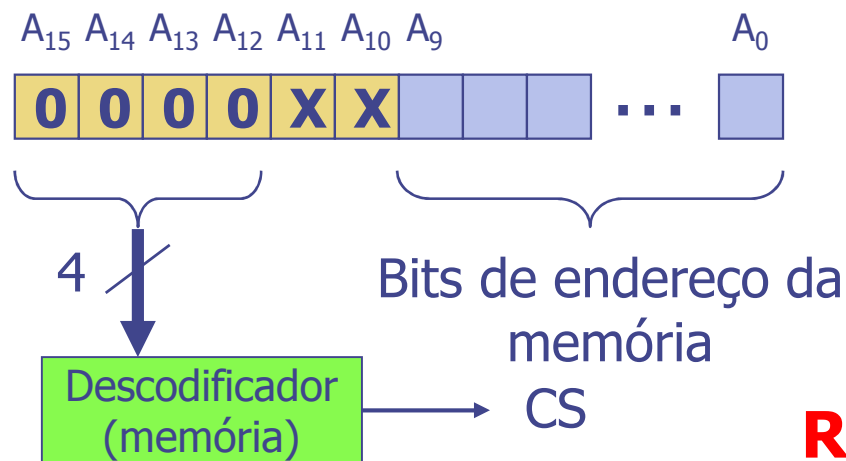


- A memória ocupa, artificialmente, 4k endereços do espaço de endereçamento quando necessita apenas de 1k (4 endereços possíveis para cada posição de memória)
- A zona ocupada não é contígua

$$CS = A_{13} \setminus . A_{12} \setminus . A_{11} \setminus . A_{10} \setminus$$

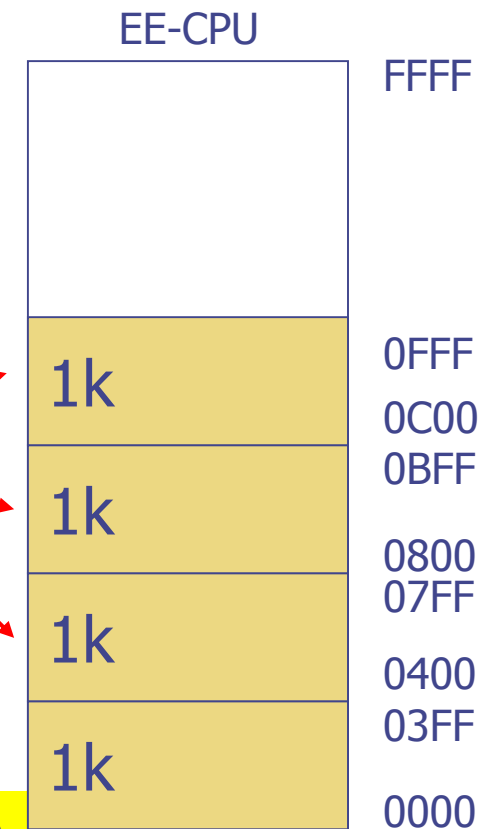
## Descodificação de endereços – descodificação parcial

- Solução 3 – usar A15, A14, A13 e A12 e ignorar A11 e A10, e.g. **0000xx**. Isto significa que um endereço válido para aceder à memória não depende do valor dos bits A11 e A10, mas tem que ter os bits A15 a A12 a 0.



- A memória ocupa, artificialmente, 4k endereços do espaço de endereçamento quando necessita apenas de 1k (4 endereços possíveis para cada posição de memória)
- Zona ocupada é contígua  $CS = A_{15} \setminus . A_{14} \setminus . A_{13} \setminus . A_{12} \setminus$

**Réplicas**



# Descodificação de endereços – exercício

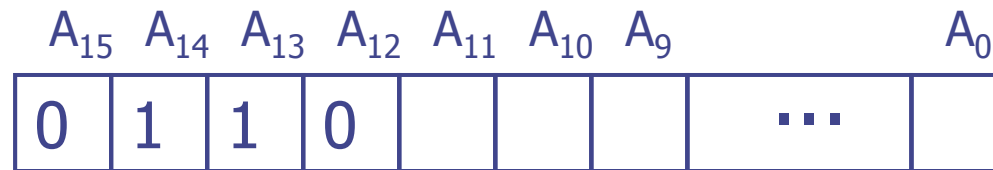
- Escrever a equação lógica do descodificador de endereços para uma memória de 4 kByte, mapeada num espaço de endereçamento de 16 bits, que respeite os seguintes requisitos:
  - Endereço-base: 0x6000; descodificação total.

$$4 \text{ kByte} = 2^{12} \quad (2^{12} - 1 = 0x0FFF)$$

$$\text{Endereço final da memória: } 0x6000 + 2^{12} - 1 = 0x6FFF$$

0110000000000000 (0x6000)

0110111111111111 (0x6FFF)



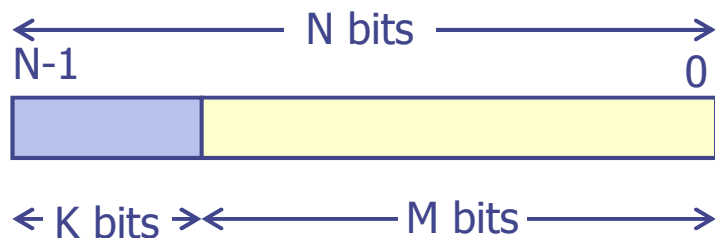
4 bits → Descodificação

12 bits → memória

- Lógica positiva:  $CS = A_{15} \cdot A_{14} \cdot A_{13} \cdot A_{12}$
- Lógica negativa:  $CS = A_{15} + A_{14} + A_{13} + A_{12}$

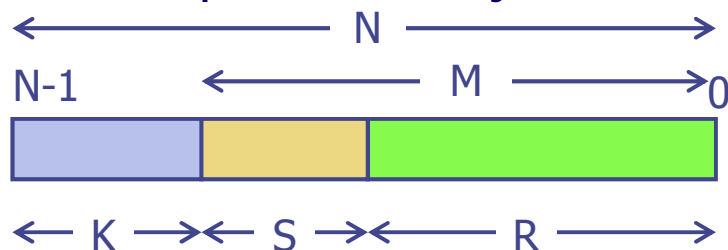
# Gerador de sinais de seleção programável

- Como se viu anteriormente, os  $N$  bits do espaço de endereçamento podem, para efeitos de descodificação de endereços e endereçamento, ser divididos em dois grupos:  $K$  bits, usados para descodificação,  $M$  bits, usados para endereçamento dentro da gama descodificada



- Dimensão da gama descodificada:  $2^M$
- Número de gamas que podem ser descodificadas:  $2^K$
- Endereço inicial da gama descodificada é definida pela combinação binária dos  $K$  bits

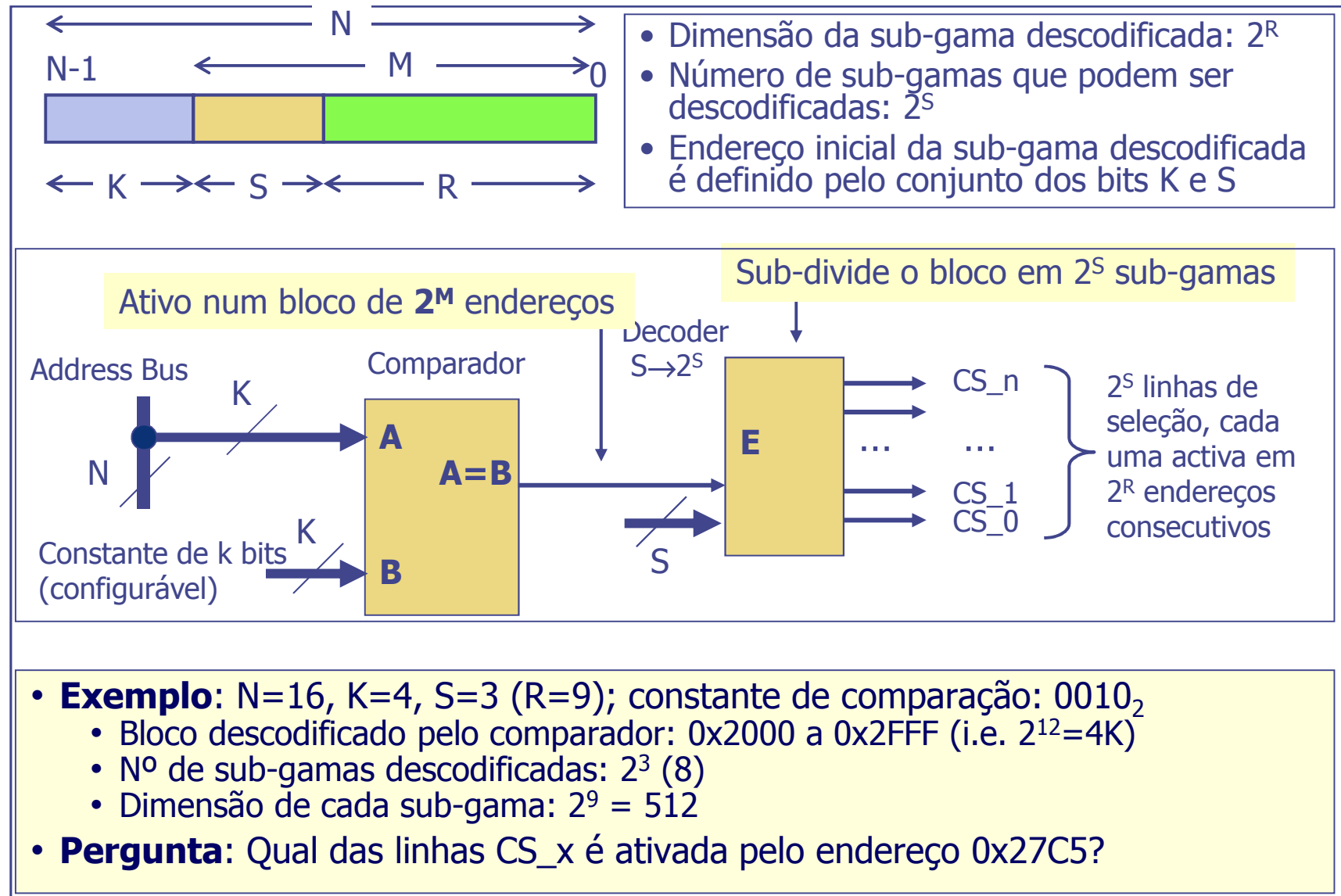
- O mesmo método pode ser aplicado para a sub-divisão dos  $M$  bits da gama descodificada:  $S$  bits usados para descodificação,  $R$  bits usados para endereçamento



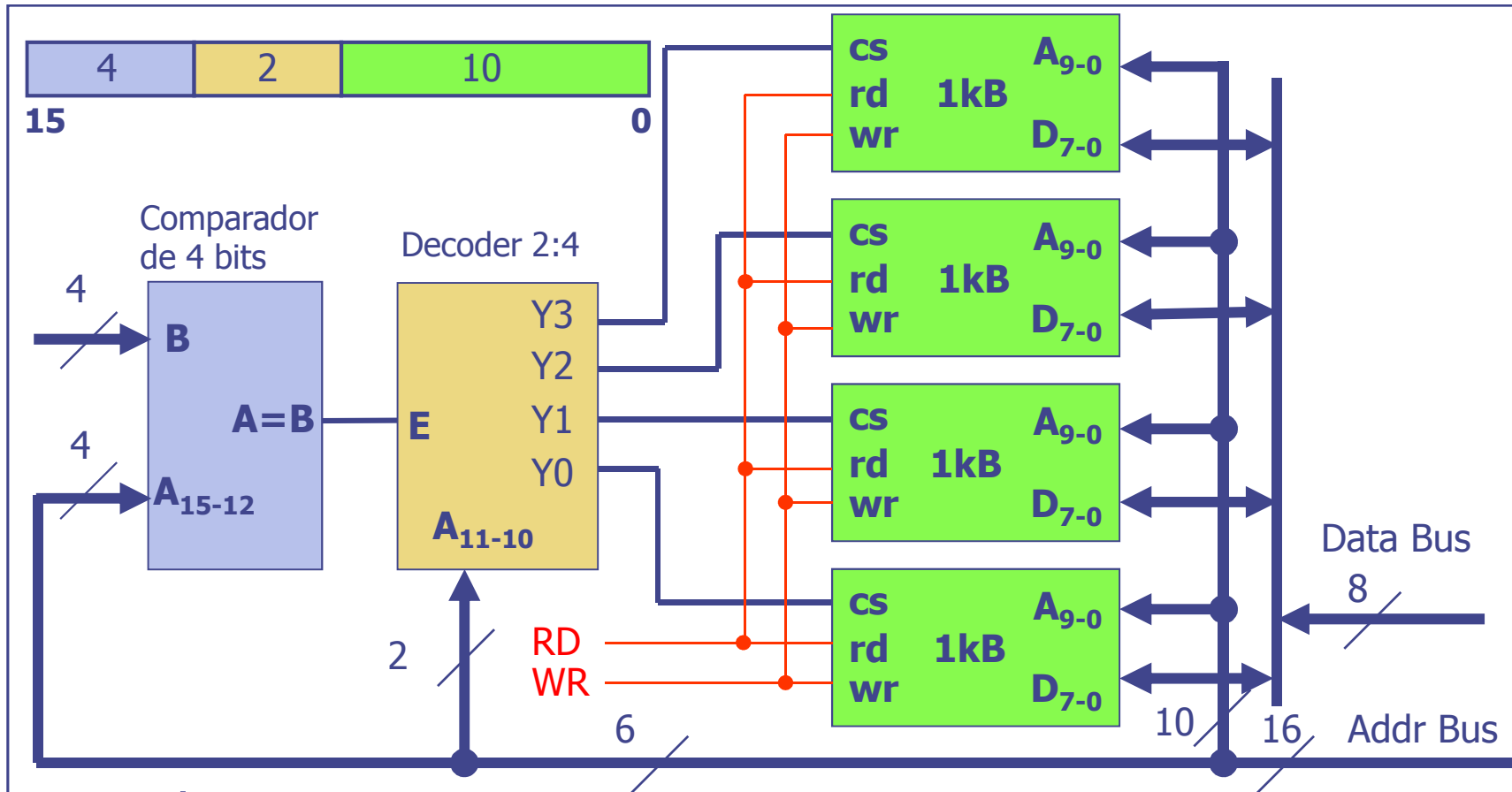
- Dimensão da sub-gama descodificada:  $2^R$
- Número de sub-gamas que podem ser descodificadas:  $2^S$
- Endereço inicial da sub-gama descodificada é definido pelo conjunto dos bits  $K$  e  $S$



# Gerador de sinais de seleção programável



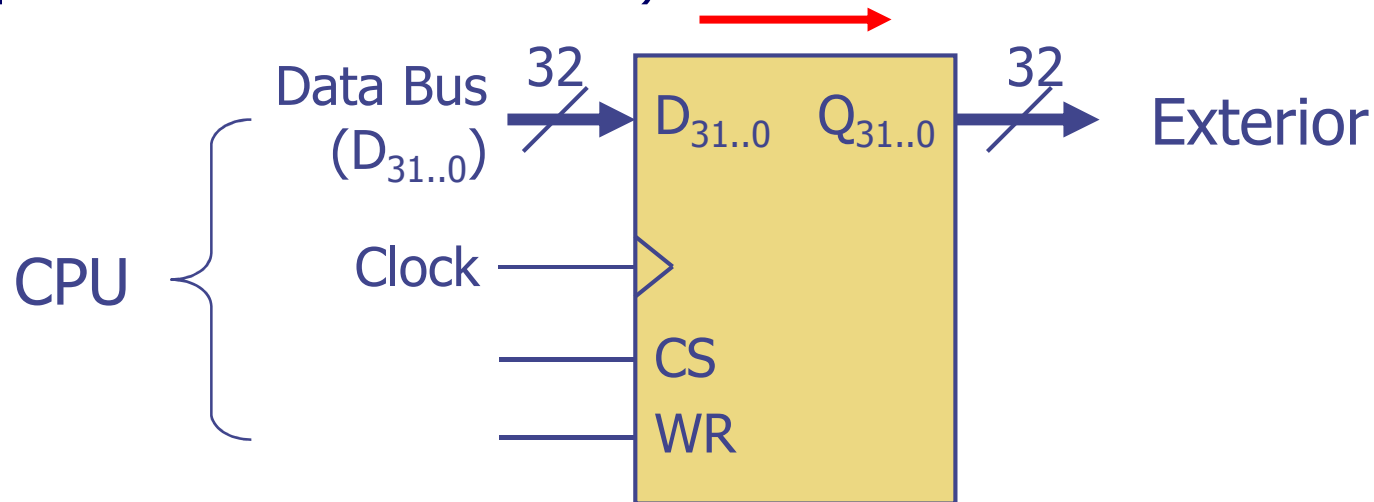
## Gerador de sinais de seleção programável - exercício



- Exercício. Com  $B=1011_2$ :
  - Indique qual a memória selecionada quando o endereço é 0xB935;
  - Construa o mapa de memória com os endereços inicial e final de cada uma das 4 memórias.

## Exemplos de portos de E/S – porto de saída de 32 bits

- Porto de saída de 32 bits (um porto de saída é um dispositivo com capacidade de armazenamento)

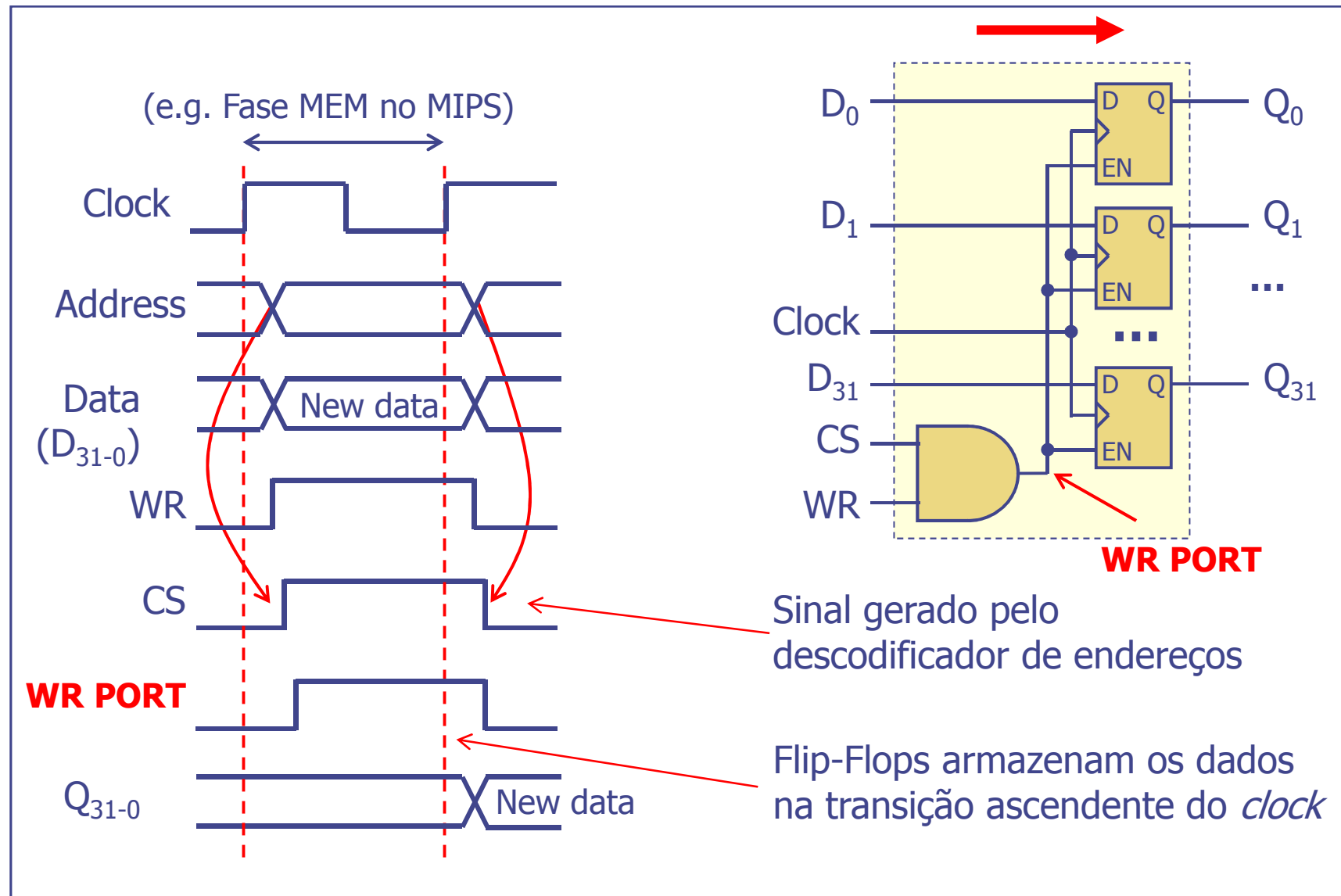


- O porto armazena informação proveniente do CPU, transferida durante uma operação de escrita na memória (fase MEM das instruções "**sw**" no caso do MIPS)
- A escrita no porto é feita na transição ativa do relógio se os sinais "**CS**" e "**WR**" estiverem ambos ativos
- O sinal "**CS**" é gerado pelo decodificador de endereços: fica ativo se o endereço gerado pelo CPU coincidir com o endereço atribuído ao porto

## Porto de saída de 32 bits (descrição em VHDL)

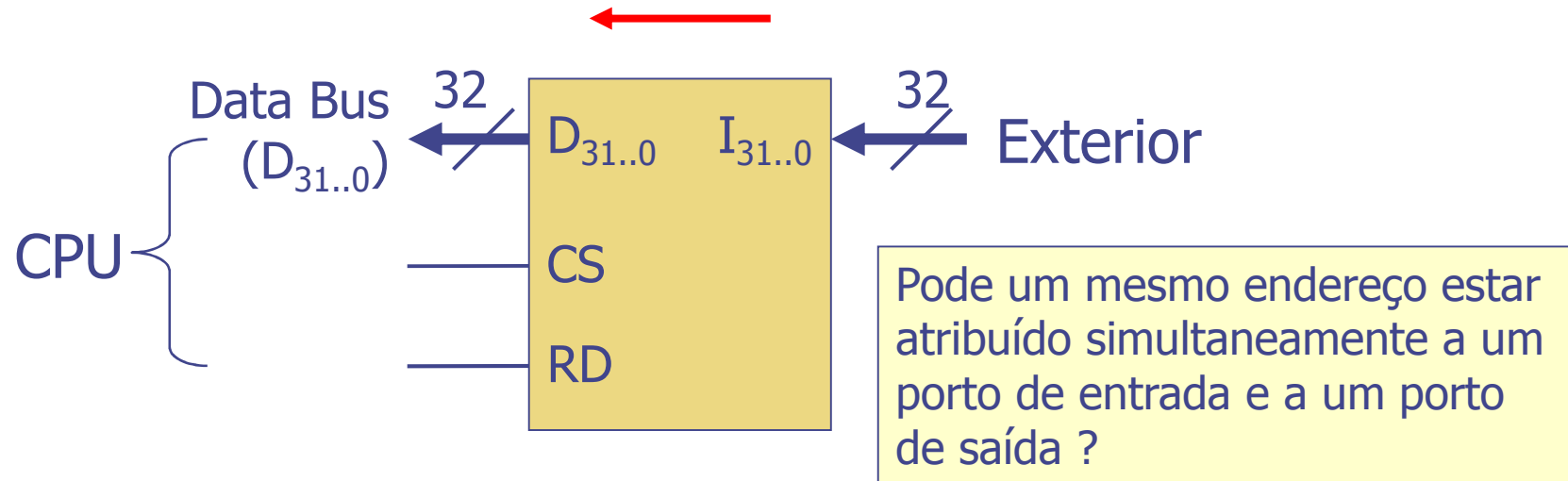
```
entity OutPort is
    port (clk, wr, cs : in  std_logic;
          dataIn      : in  std_logic_vector(31 downto 0);
          dataOut      : out std_logic_vector(31 downto 0));
end OutPort;
architecture behav of OutPort is
begin
    process (clk)
    begin
        if (rising_edge(clk)) then
            if (cs = '1' and wr = '1') then
                dataOut <= dataIn;
            end if;
        end if;
    end process;
end behav;
```

# Porto de saída de 32 bits



## Exemplos de portos de E/S – porto de entrada de 32 bits

- Porto de entrada de 32 bits (em geral, um porto de entrada não tem capacidade de armazenamento)

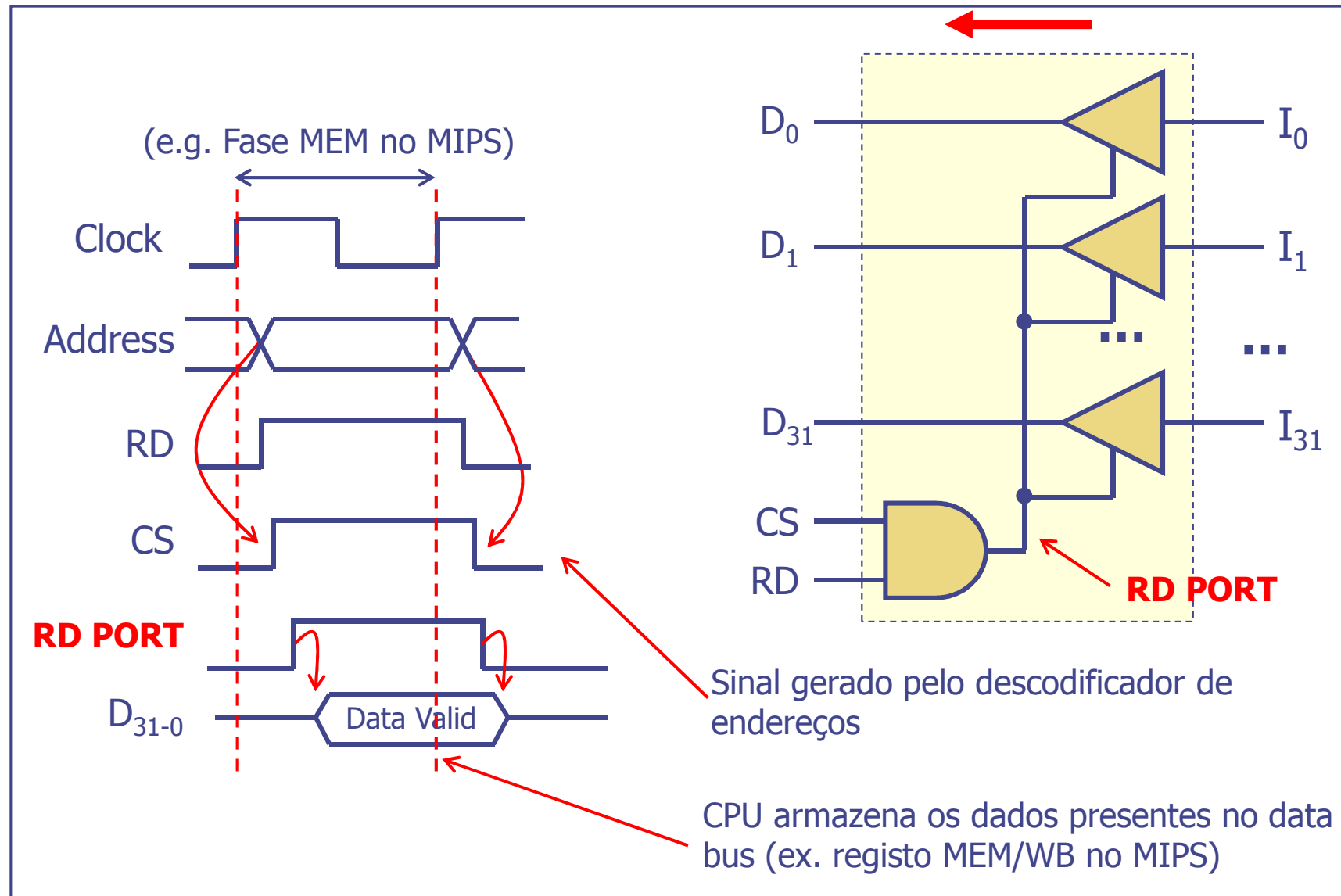


- A informação presente nas 32 linhas de entrada ( $I_{31..0}$ ) é transferida para o CPU durante uma operação de leitura (fase MEM das instruções "**lw**", no caso do MIPS)
- As saídas  $D_{31..0}$  têm obrigatoriamente portas *tri-state* que só são ativadas quando estão ativos, simultaneamente, os sinais "**CS**" e "**RD**"
- Ao nível do porto, a operação de leitura é assíncrona, pelo que não é necessário o sinal de relógio

# Porto de entrada (descrição em VHDL)

```
entity InPort is
    port(rd, cs : in std_logic;
          dataIn : in std_logic_vector(31 downto 0);
          dataOut : out std_logic_vector(31 downto 0));
end InPort;
architecture behav of InPort is
begin
    process(rd, cs, dataIn)
    begin
        if(cs = '1' and rd = '1') then
            dataOut <= dataIn;
        else
            dataOut <= (others => 'Z');
        end if;
    end process;
end behav;
```

# Porto de entrada de 32 bits



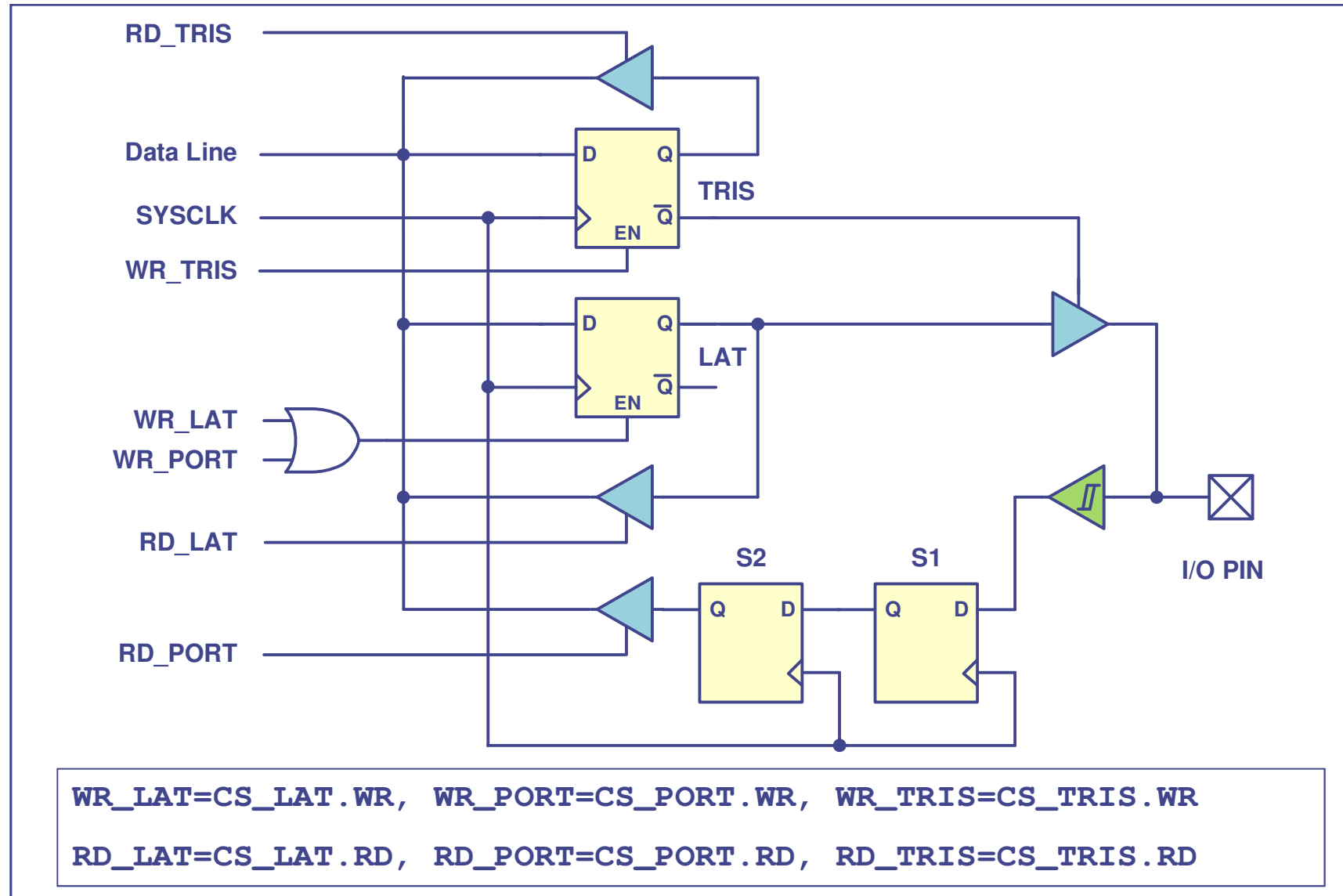


# Portos de I/O no PIC32

- O microcontrolador PIC32MX795F512H disponibiliza vários portos de I/O, com várias dimensões (16 bits, no máximo), identificados com as siglas:
  - RB (16 bits, I/O)
  - RC (1 bit, I/O)
  - RD (8 bits, I/O)
  - RE (4 bits, I/O)
  - RF (5 bits, I/O)
  - RG (4 de I/O + 2 I)
- Cada um dos bits de cada um destes portos pode ser configurado, por programação, como entrada ou saída
- Ou seja, **um porto de I/O de n bits do PIC32 é um conjunto de n portos de I/O de 1 bit**

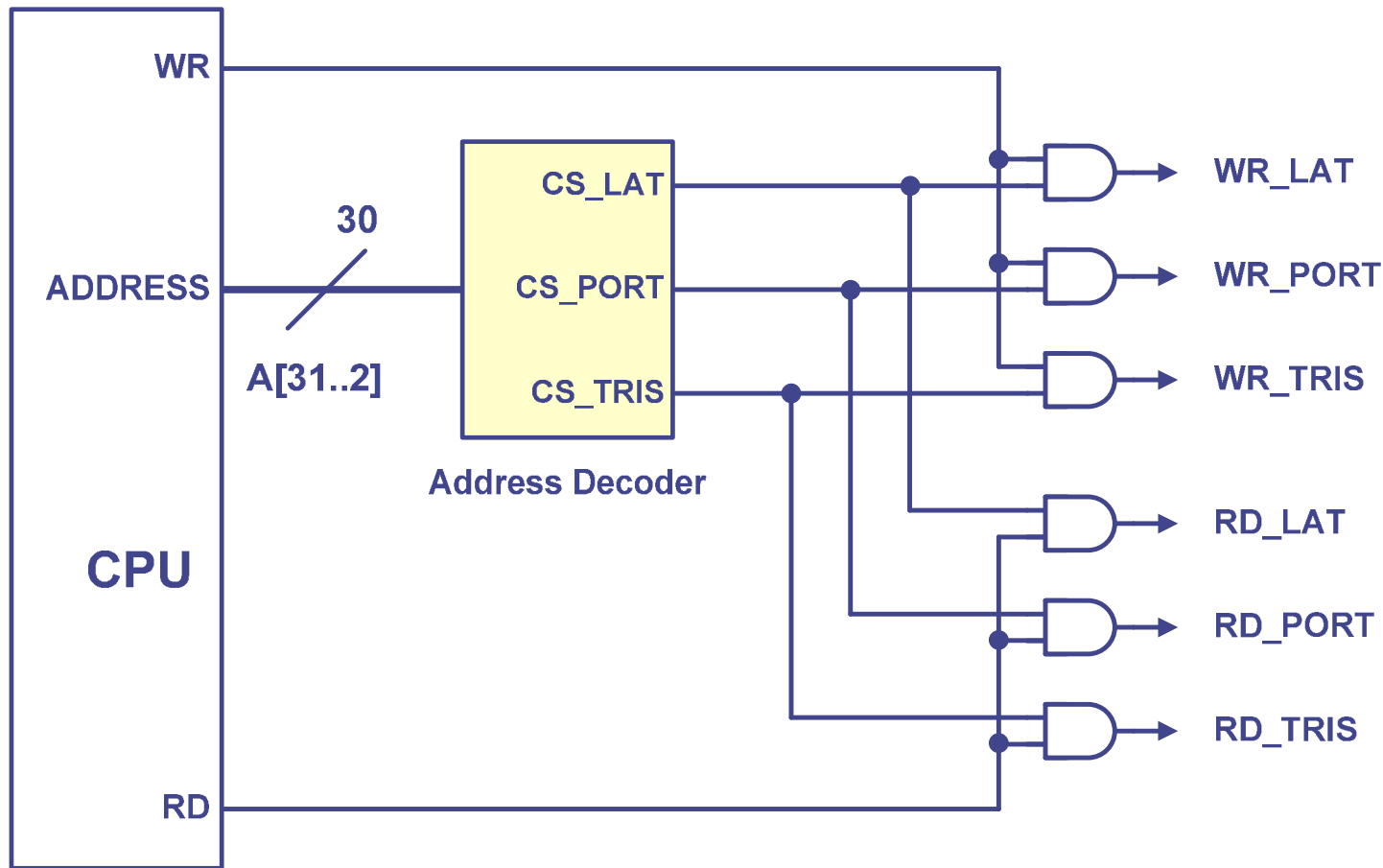
# Portos de I/O no PIC32

- Cada um dos portos (B a G) tem associado um total de 12 registros de 32 bits. Desses, os que vamos usar são:
  - **TRIS** – usado para configuração do porto (entrada ou saída)
  - **PORT** – usado para ler valores de um porto de entrada
  - **LAT** – usado para escrever valores num porto de saída
- A configuração de cada um dos bits de um porto, como entrada ou saída, é feita através dos registros **TRISx** ("Tri-state" *registers*), em que **x** é a letra que identifica o porto (B a G)
  - **TRISxn = 1**, bit **n** do porto **x** configurado como entrada
  - **TRISxn = 0**, bit **n** do porto **x** configurado como saída

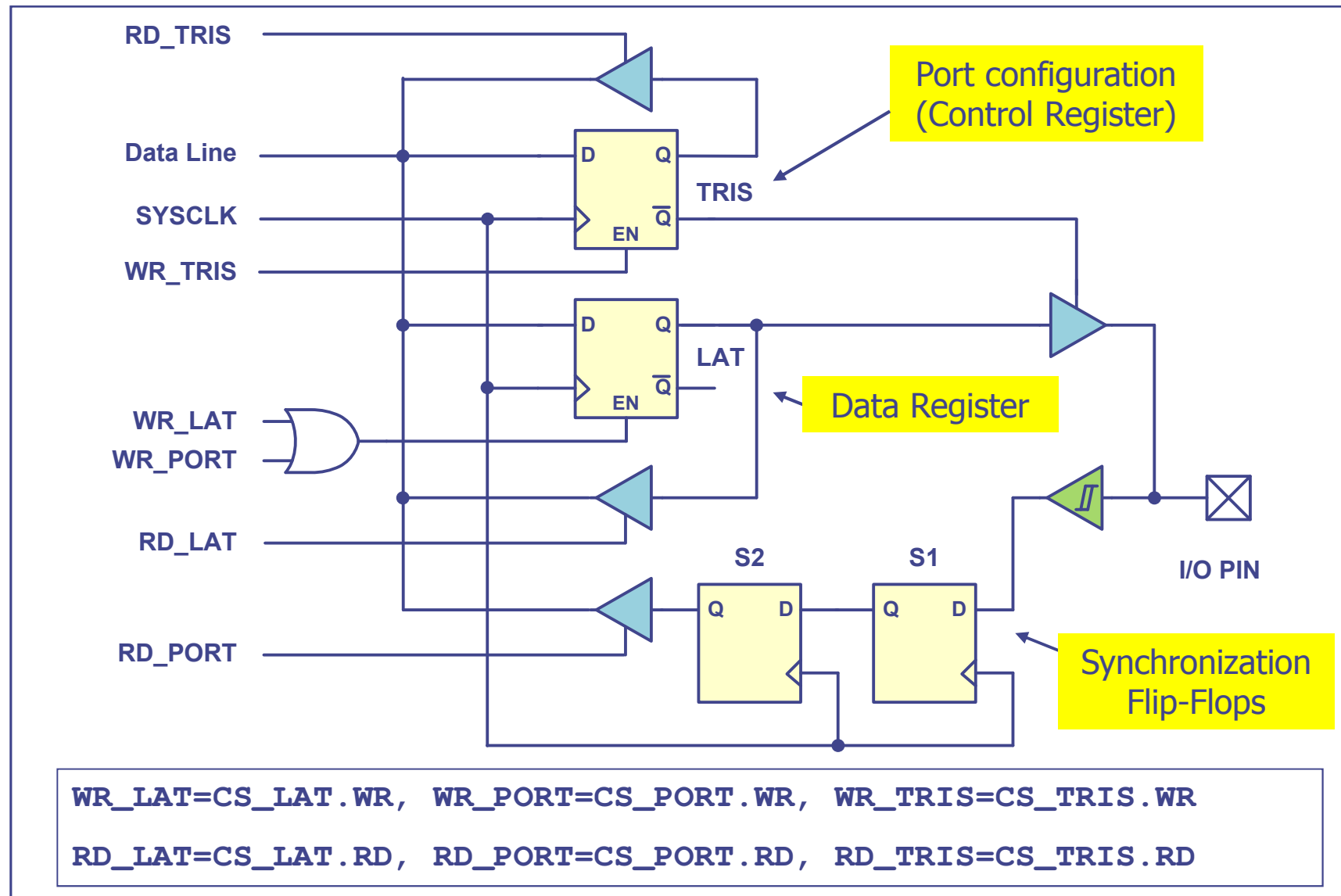


## Descodificação de endereços para acesso aos portos

- Os sinais que permitem a escrita e a leitura dos 3 registos de um porto (TRIS, PORT e LAT) podem ser obtidos do seguinte modo:



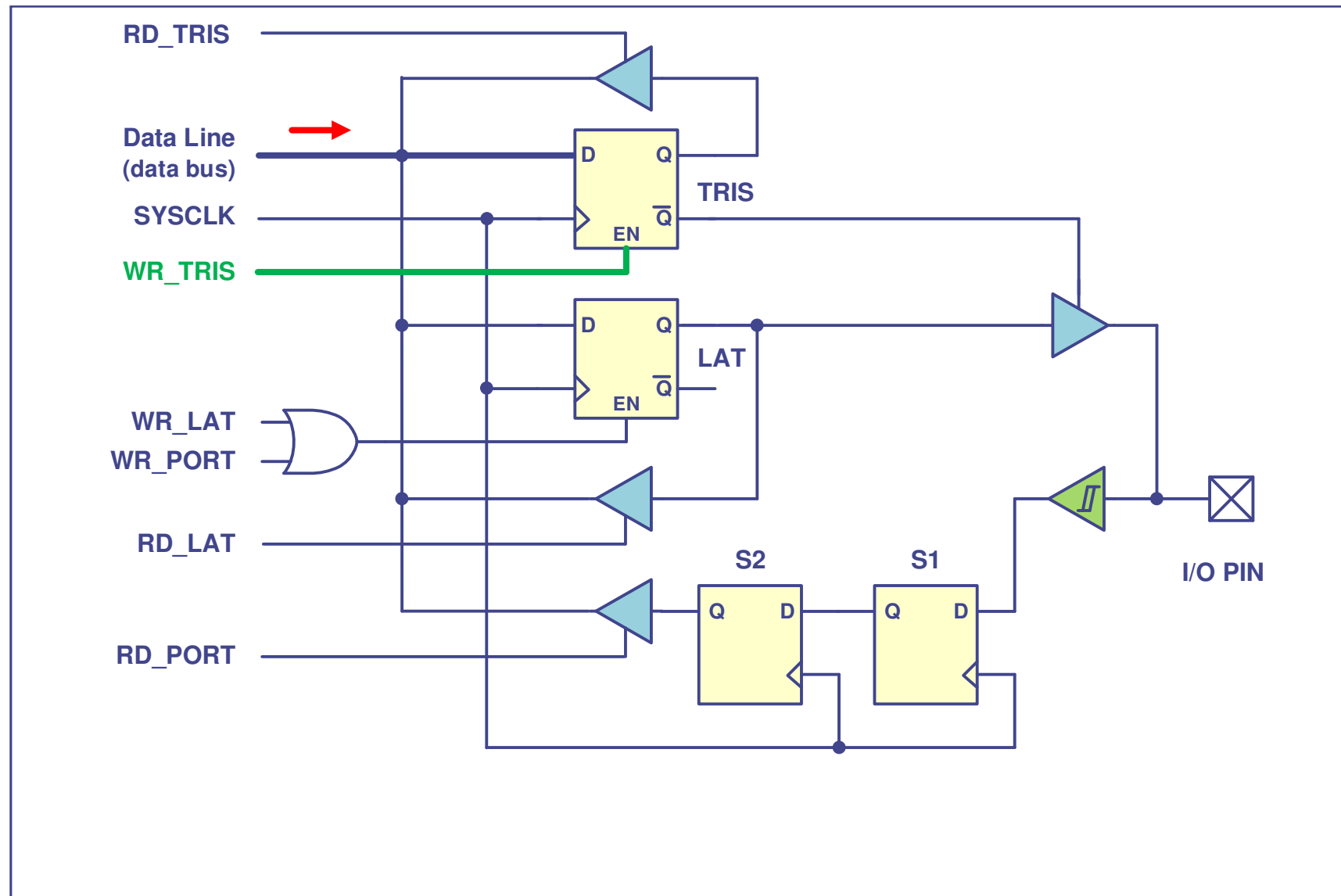
# Modelo simplificado de um porto de I/O de 1 bit no PIC32



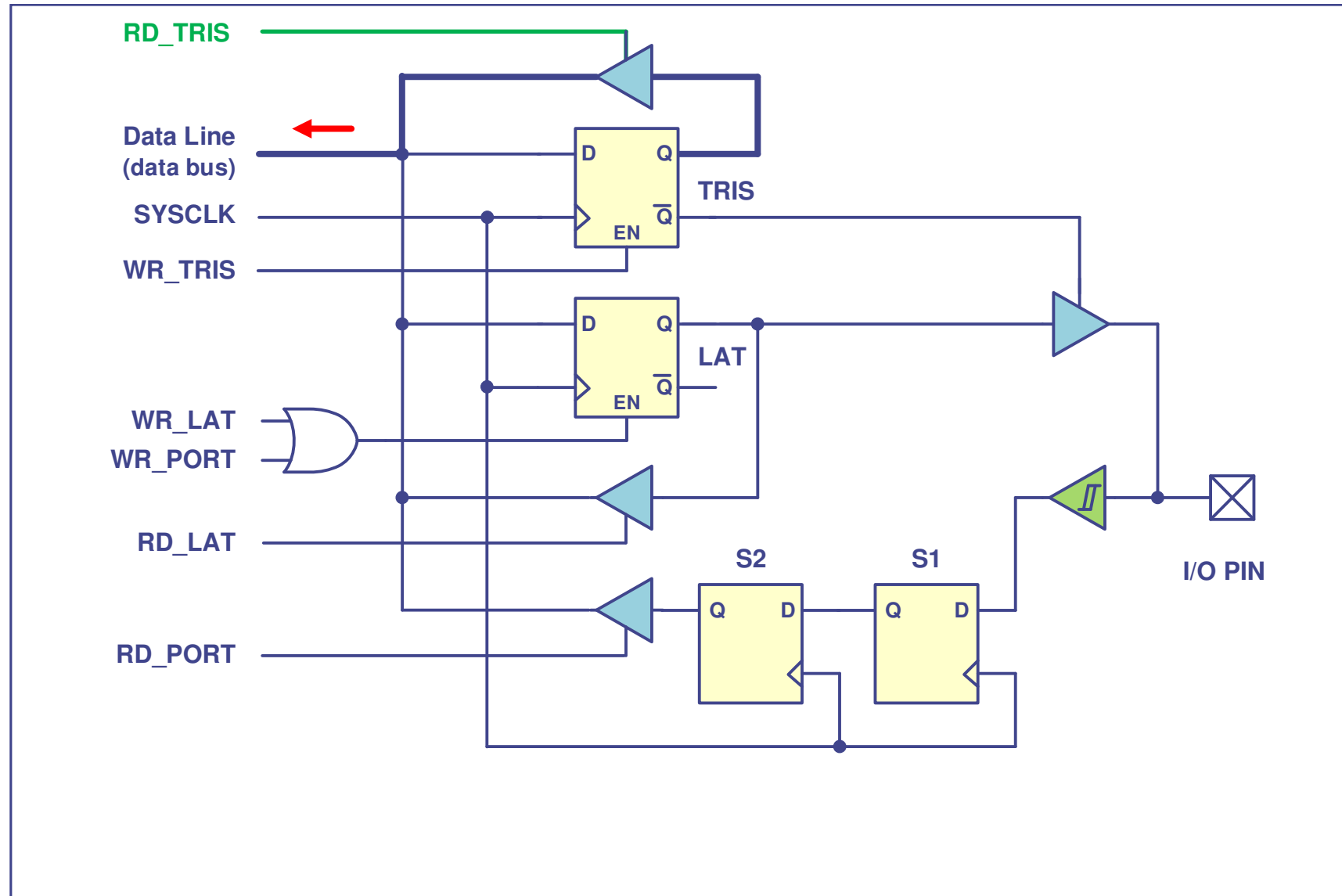
# Portos de I/O no PIC32

- A escrita no porto é efetuada no endereço referenciado pelo identificador **LATx**, em que x é a letra que identifica o porto
- A leitura do porto é feita do endereço referenciado por **PORTx**
- Os portos estão mapeados no espaço de endereçamento unificado do PIC32 (ver aula 2), em endereços definidos pelo fabricante
- Exemplos:   Endereço de **TRISB**:   **0xBF886040**  
                  Endereço de **PORTB**:   **0xBF886050**  
                  Endereço de **LATB**:     **0xBF886060**
- Cada porto de entrada inclui uma porta *schmitt trigger* (comparador com histerese) que tem o objetivo de melhorar a imunidade ao ruído
- No porto de entrada, o sinal externo é sincronizado através de 2 *flip-flops*. Esta configuração visa resolver os possíveis problemas causados por meta-estabilidade decorrentes do facto de o sinal externo ser assíncrono relativamente ao *clock* do CPU
- Os dois *flip-flops*, em conjunto, impõem um atraso de, até, dois ciclos de relógio na propagação do sinal até ao barramento de dados do CPU

# Modelo simplificado de um porto de I/O de 1 bit no PIC32

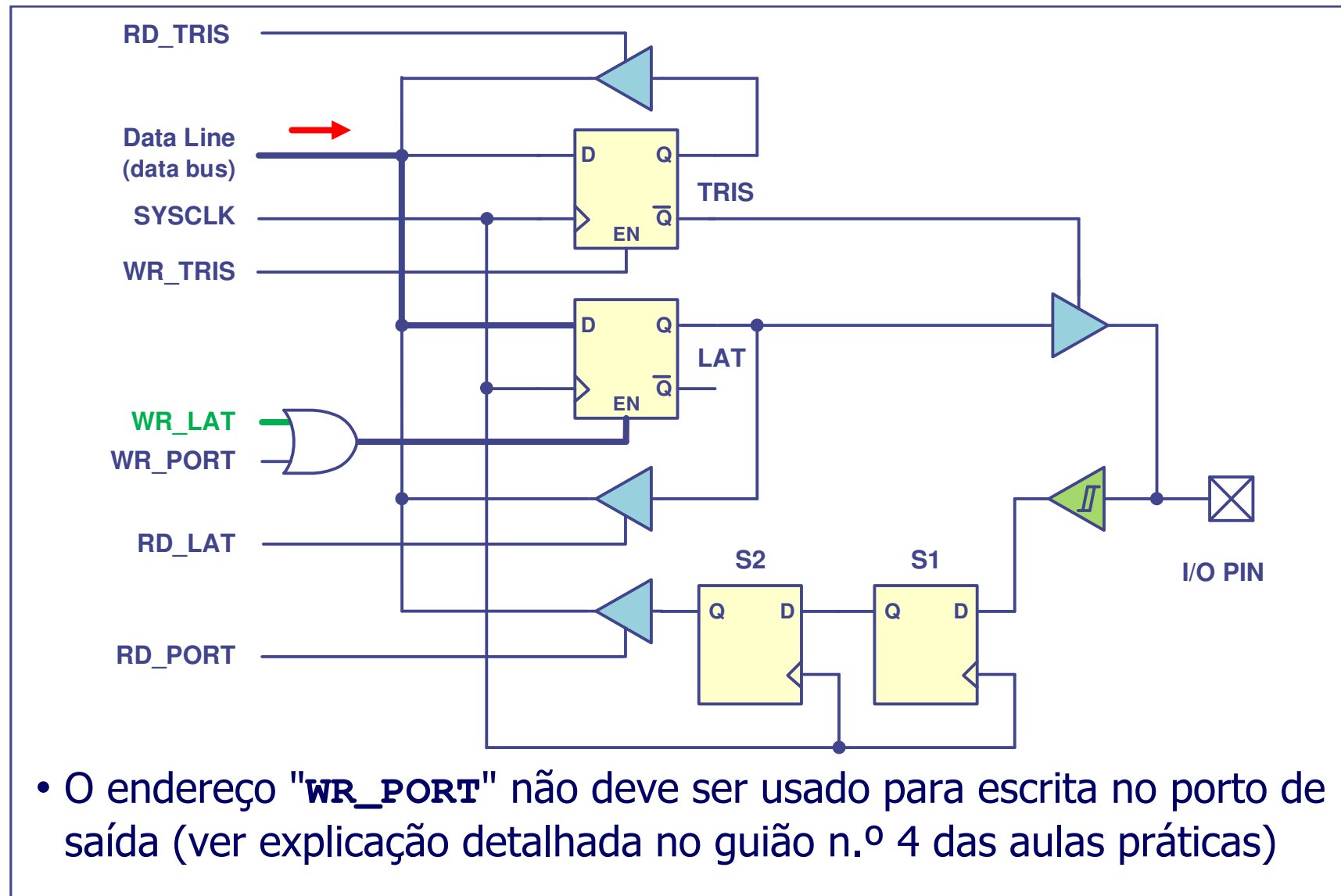


# Modelo simplificado de um porto de I/O de 1 bit no PIC32

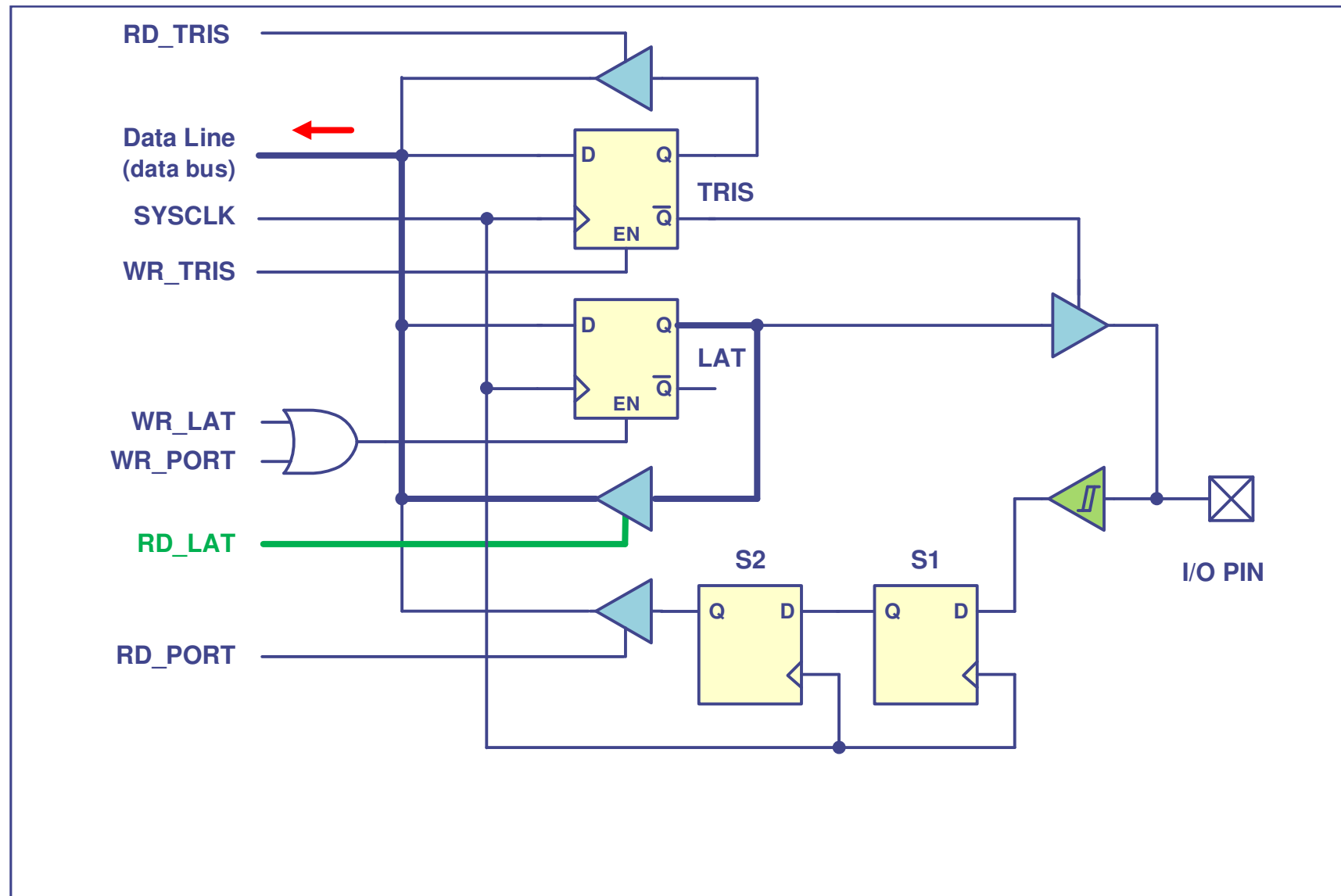




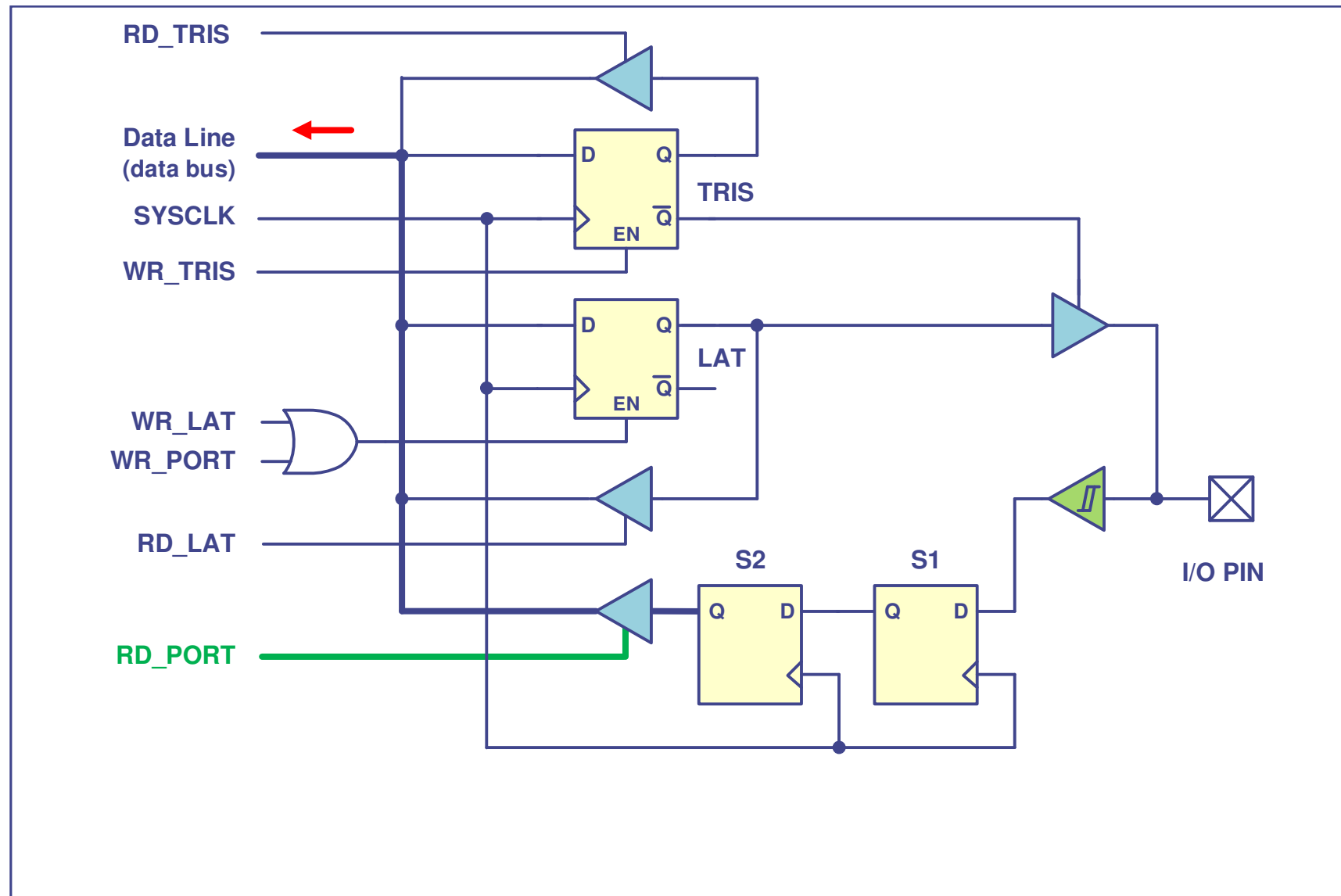
## Modelo simplificado de um porto de I/O de 1 bit no PIC32



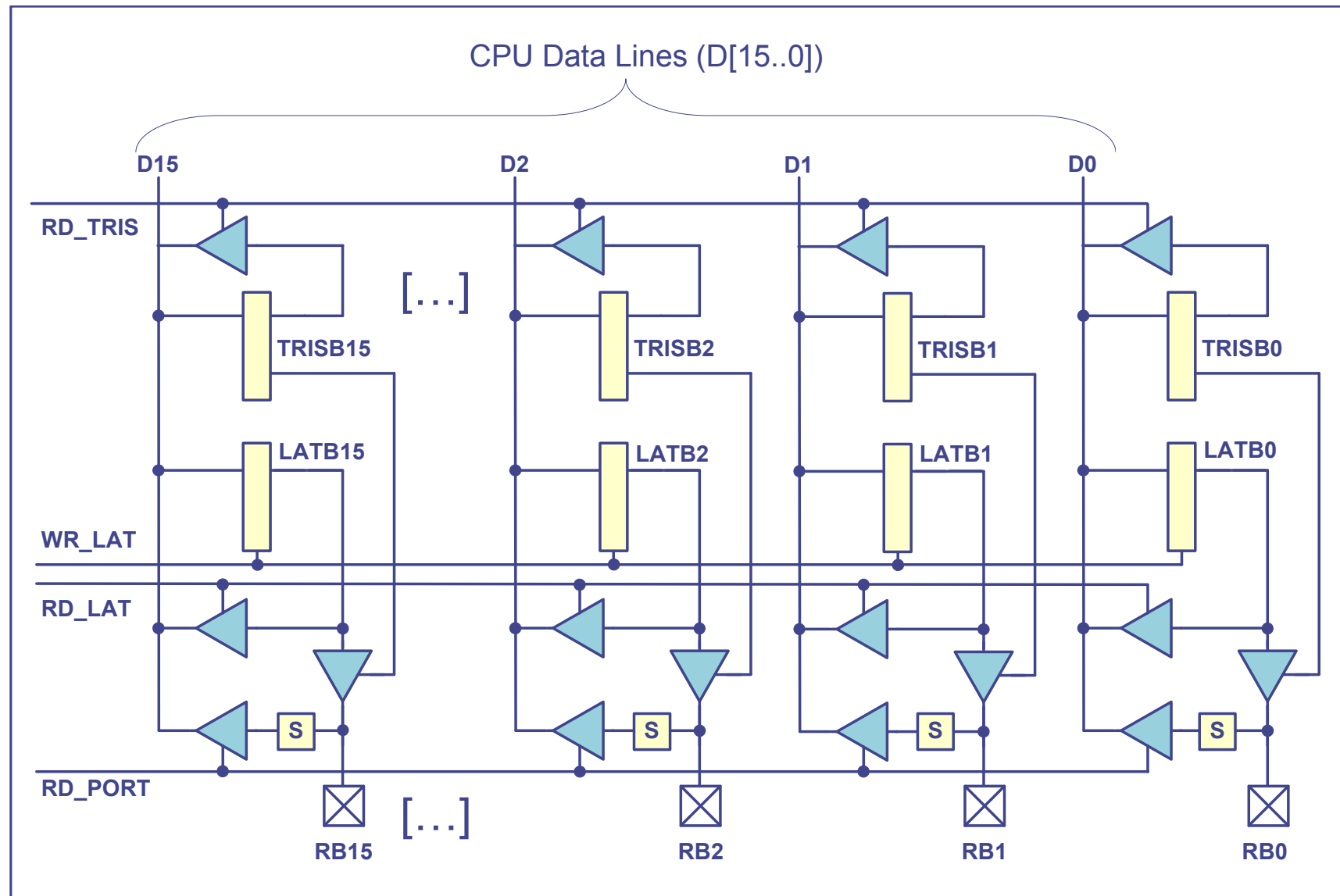
# Modelo simplificado de um porto de I/O de 1 bit no PIC32



# Modelo simplificado de um porto de I/O de 1 bit no PIC32



# Modelo simplificado de um porto de I/O no PIC32



# Exemplo de configuração/utilização dos portos

- Definição dos endereços dos portos:

```
.equ  SFR_BASE_HI, 0xBF88

.equ  TRISD, 0x60C0 # TRISD address is 0xBF8860C0
.equ  PORTD, 0x60D0 # TRISD address is 0xBF8860D0
.equ  LATD, 0x60E0 # LATD address is 0xBF8860E0

.equ  TRISE, 0x6100 # TRISE address is 0xBF886100
.equ  PORTE, 0x6110 # PORTE address is 0xBF886110
.equ  LATE, 0x6120 # LATE address is 0xBF886120

.data

.text
.globl main
```

# Exemplo de configuração/utilização dos portos

- Gerar no bit 0 do porto D (RD0) um sinal de 1 Hz com *duty-cycle* de 10% (i.e. RD0=1 durante 0.1s, RD0=0 durante 0.9s)

```
.text
.globl  main
main: lui  $t0, SFR_BASE_HI    # 16 MSbits of port addresses
      lw   $t1, TRISD($t0)     # Read TRISD register
      andi $t1, $t1, 0xFFFE    # Modify bit 0 (0 is OUT)
      sw   $t1, TRISD($t0)     # Write TRISD (port configured)
loop: lw   $t1, LATD($t0)      # Read LATD
      ori  $t1, $t1, 0x0001    # Modify bit 0 (set)
      sw   $t1, LATD($t0)      # Write LATD
# wait 100 ms (e.g., using MIPS core timer)
      lw   $t1, LATD($t0)      # Read LATD
      andi $t1, $t1, 0xFFFE    # Modify bit 0 (reset)
      sw   $t1, LATD($t0)      # Write LATD
# wait 900 ms (e.g., using MIPS core timer)
      j    loop
```

# Exemplo de configuração/utilização dos portos

- Configurar os bit 0 e 1 do porto E (RE0 e RE1) como entrada e saída, respetivamente. Em ciclo infinito, ler o valor do porto de entrada e escrever esse valor, negado, no porto de saída

```
.text
.globl  main
main: lui  $t0, SFR_BASE_HI  # 16 MSbits of port addresses
      lw   $t1, TRISE($t0)    # Read TRISE register
      ori  $t1, $t1, 0x0001   # bit0 = 1 (IN)
      andi $t1, $t1, 0xFFFFD  # bit1 = 0 (OUT)
      sw   $t1, TRISE($t0)    # TRISE configured

loop: lw   $t1, PORTE($t0)    # Read PORTE register
      sll  $t2, $t1, 1        #
      andi $t2, $t2, 0x0002    # Reset all bits except bit 1
      xori $t2, $t2, 0x0002    # Negate bit 1
      andi $t1, $t1, 0xFFFFD  # Reset bit 1 in port value
      or   $t1, $t1, $t2      # Merge data
      sw   $t1, LATE($t0)     # Write LATE register
      j    loop
```

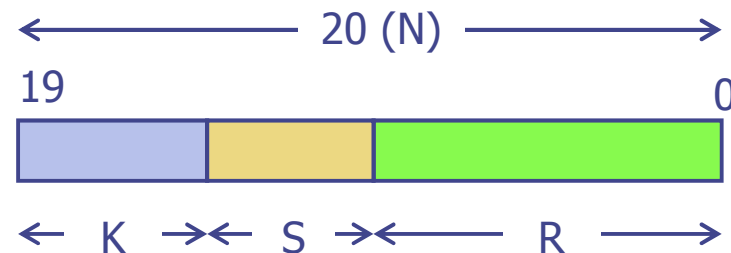
# Descodificação de endereços - exercícios

1. Para o exemplo do slide 27, suponha que no decodificador apenas se consideram os bits A15, A13 e A11, com os valores 1, 0 e 0, respetivamente.
  - a) Apresente a expressão lógica que implementa este decodificador: i) em lógica positiva e ii) em lógica negativa.
  - b) Indique os endereços inicial e final da gama-base decodificada e de todas as réplicas.
2. Suponha que, no exercício do slide 31, não se decodificaram os bits A14 e A12, resultando na expressão  $CS\_x = A15 + A13$ 
  - a) Indique as gamas do espaço de endereçamento de 16 bits ocupadas pela memória.
  - b) Indique os endereços possíveis para aceder à 15ª posição da memória.
3. Escreva as equações lógicas dos 4 decodificadores necessários para a geração dos sinais de seleção para cada um dos dispositivos do exemplo do slide 16.
4. Para o exemplo do slide 33, determine a gama de endereços em que cada uma das linhas CS\_x está ativa, com a constante de comparação  $0010_2$



# Gerador de sinais de seleção programável - exercícios

1. Pretende-se gerar os sinais de seleção para 4 memórias de 8 kByte, a mapear em gamas de endereços consecutivas, de modo a formar um conjunto de 32 kByte. O endereço inicial deve ser configurável. Para um espaço de endereçamento de 20 bits:



- Indique o número de bits dos campos K, S e R, supondo descodificação total.
- Esboce o circuito digital que implementa este descodificador
- Indique os endereços inicial e final para a primeira, segunda e última gamas de endereços possíveis de serem descodificadas.
- Para a última gama de endereços, indique os endereços inicial e final atribuídos a cada uma das 4 memórias de 8k
- Suponha que o endereço 0x3AC45 é um endereço válido para aceder ao conjunto de 32k. Indique os endereços inicial e final da gama que inclui este endereço. Indique os endereços inicial e final da memória de 8K à qual está atribuído este endereço

# Gerador de sinais de seleção programável - exercícios

1. Pretende-se gerar os sinais de seleção para os seguintes 4 dispositivos: 1 porto de saída de 1 byte, 1 memória RAM de 1 kByte (*byte-addressable*), 1 memória ROM de 2 kByte (*byte-addressable*), 1 periférico com 5 registos de 1 byte cada um. O espaço de endereçamento a considerar é de 20 bits.
  - a) Desenhe o gerador de linhas de seleção para estes 4 dispositivos, baseando-se no modelo discutido nos slides anteriores e usando a mesma sub-gama para o periférico e para o porto de saída de 1 byte.
  - b) Especifique a dimensão de todos os barramentos e quais os bits que são usados.
  - c) Desenhe o mapa de memória com o endereço inicial e final do espaço efetivamente ocupado por cada um dos 4 dispositivos, considerando para o conjunto um endereço-base por si determinado.
2. O periférico com 5 registos, do exercício anterior, tem um barramento de endereços com três bits. Suponha que esses bits estão ligados aos bits A0, A1 e A2 do barramento de endereços do CPU.
  - a) Usando o decodificador desenhado no exercício anterior, indique os 16 primeiros endereços em que é possível aceder ao registo 0 (selecionado com A0, A1 e A2 a 0)
  - b) Repita o exercício anterior supondo que os 3 bits do barramento de endereços do periférico estão ligados aos bits A2, A3 e A4 do barramento de endereços.

# Programação de portos I/O - exercício

1. Pretende usar-se o porto RB do microcontrolador PIC32MX795F512H para realizar a seguinte função (em ciclo fechado):

O byte menos significativo ligado a este porto é lido com uma periodicidade de 100ms. Com um atraso de 10ms, o valor lido no byte menos significativo é colocado, em complemento para 1, no byte mais significativo desse mesmo porto. Escreva, em *assembly* do MIPS, um programa que execute esta tarefa.

- a) configure o porto RB para executar corretamente a tarefa descrita
- b) efetue a leitura do porto indicado
- c) execute um ciclo de espera de 10ms
- d) efetue a transformação da informação lida para preparar o processo de escrita naquela porto
- e) efetue, no byte mais significativo, o valor resultante da operação anterior
- f) execute um ciclo de espera de 90ms
- g) regresse ao ponto b)