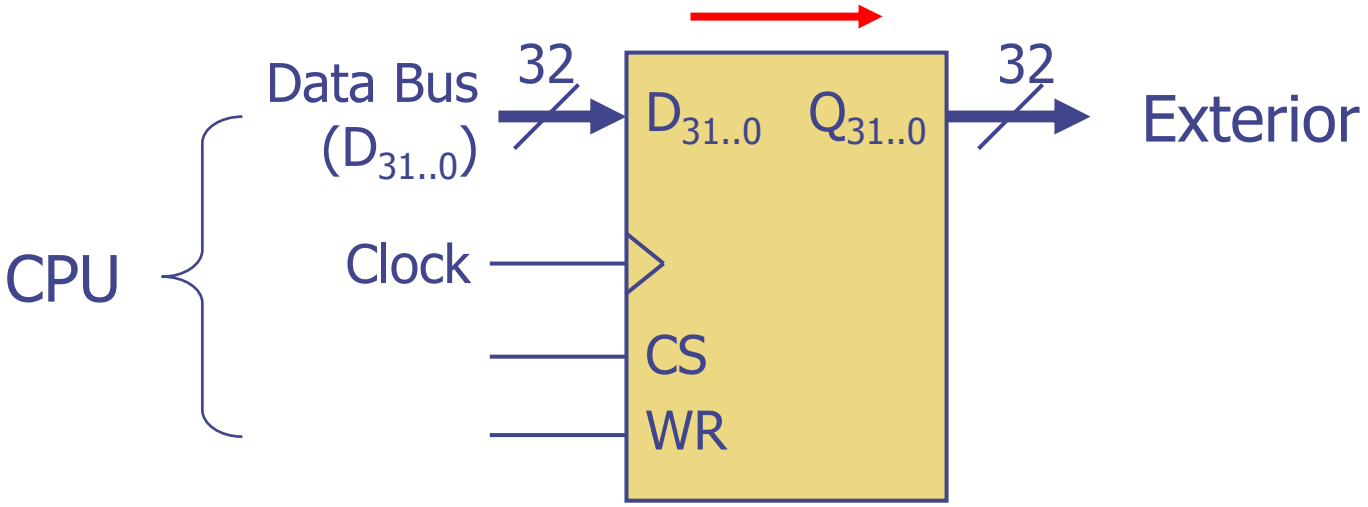
Aula 2

- Exemplos de periféricos simples:
  - Estrutura básica de portos de I/O
- Estudo de caso: portos de I/O no PIC32
  - Estrutura de um porto de I/O de 1 bit
  - Estrutura dos portos de I/O de "n" bits
  - Exemplos de programação em *assembly*
- Transferência de informação entre os periféricos e a memória: E/S programada (*polling*)

José Luís Azevedo, Bernardo Cunha, Tomás O. Silva, P. Bartolomeu

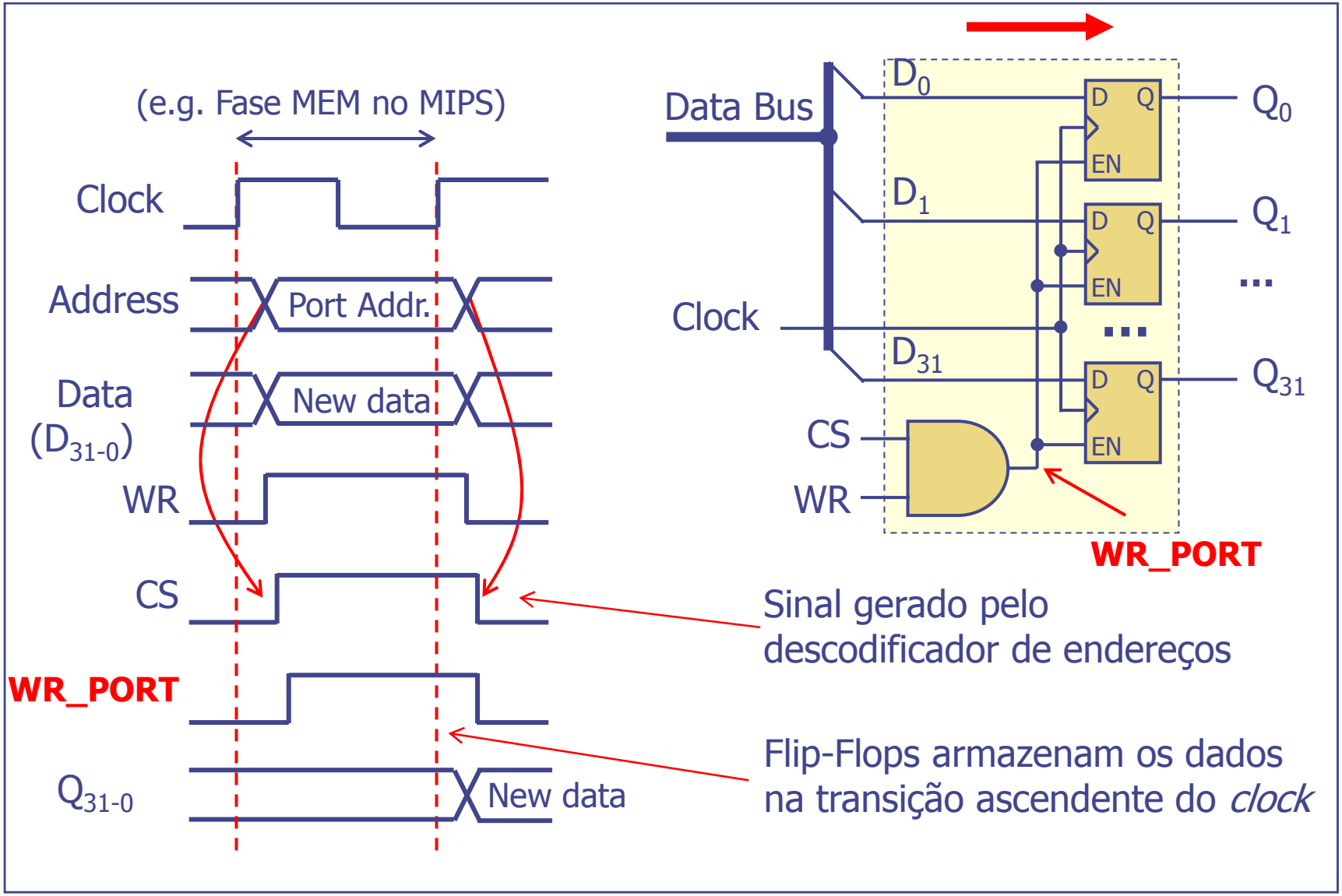
Exemplo de periférico simples: porto de saída de 32 bits

- Porto de saída de 32 bits (constituído por um único registo de 32 bits)



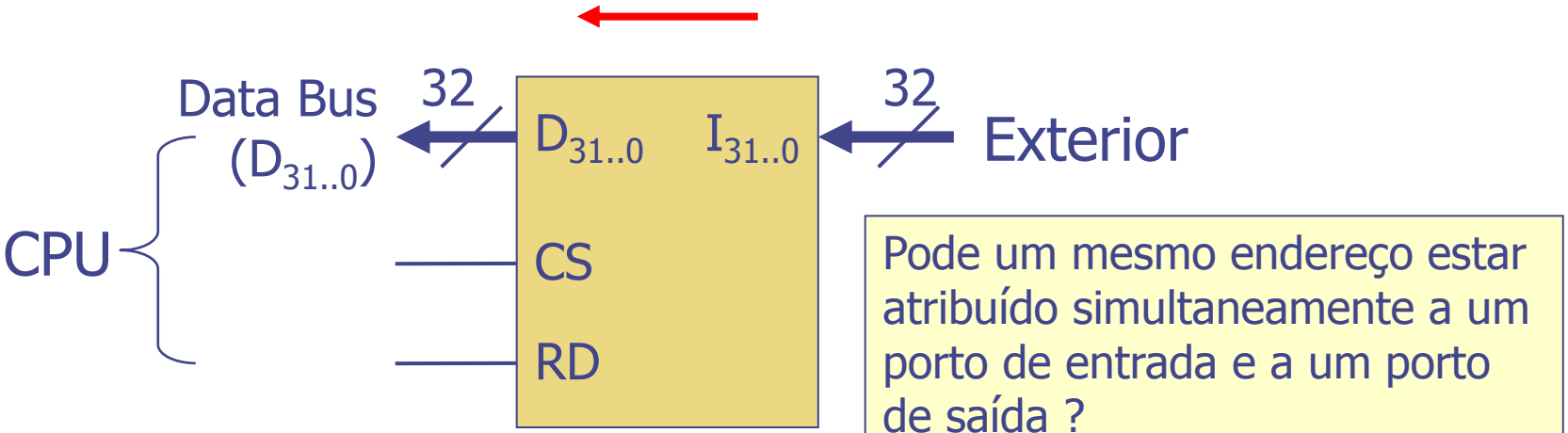
- O porto armazena informação proveniente do CPU, transferida durante uma operação de escrita na memória (estágio MEM nas instruções "**`sw`**", no caso do MIPS)
- A escrita no porto é feita na transição ativa do relógio se os sinais "**`CS`**" e "**`WR`**" estiverem ambos ativos
- O sinal "**`CS`**"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

Porto de saída de 32 bits



Exemplo de periférico simples: porto de entrada de 32 bits

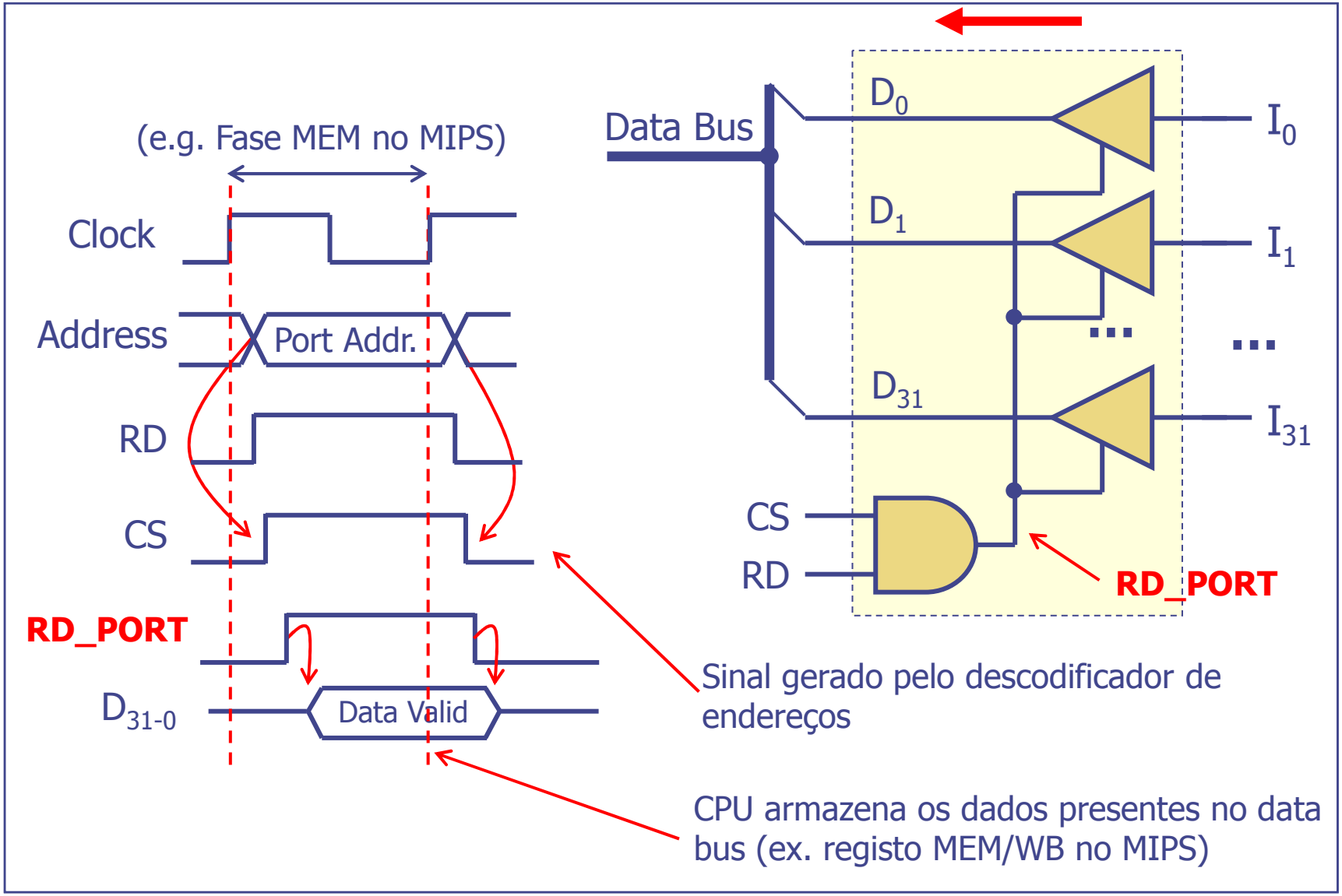
- Porto de entrada de 32 bits (em geral, um porto de entrada não tem capacidade de armazenamento)



Pode um mesmo endereço estar atribuído simultaneamente a um porto de entrada e a um porto de saída ?

- A informação presente nas 32 linhas de entrada ($I_{31..0}$) é transferida para o CPU durante uma operação de leitura (estágio MEM nas 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
- O sinal "**`CS`**" é gerado pelo descodificador de endereços: fica ativo se o endereço gerado pelo CPU coincidir com o endereço atribuído ao porto

Porto de entrada de 32 bits



- Portos de I/O no PIC32
  - Estrutura de um porto de I/O de 1 bit
  - Estrutura dos portos de I/O de "n" bits
  - Exemplos de programação em *assembly*

Estudo de caso: portos de I/O no PIC32

- O microcontrolador PIC32MX795F512H disponibiliza vários portos de I/O, com várias dimensões (16 bits, no máximo)
  - Porto B (RB): 16 bits, I/O
  - Porto C (RC): 2 bit, I/O
  - Porto D (RD): 12 bits, I/O
  - Porto E (RE): 8 bits, I/O
  - Porto F (RF): 5 bits, I/O
  - Porto G (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
  - **um porto de I/O de n bits do PIC32 é um conjunto de n portos de I/O de 1 bit**

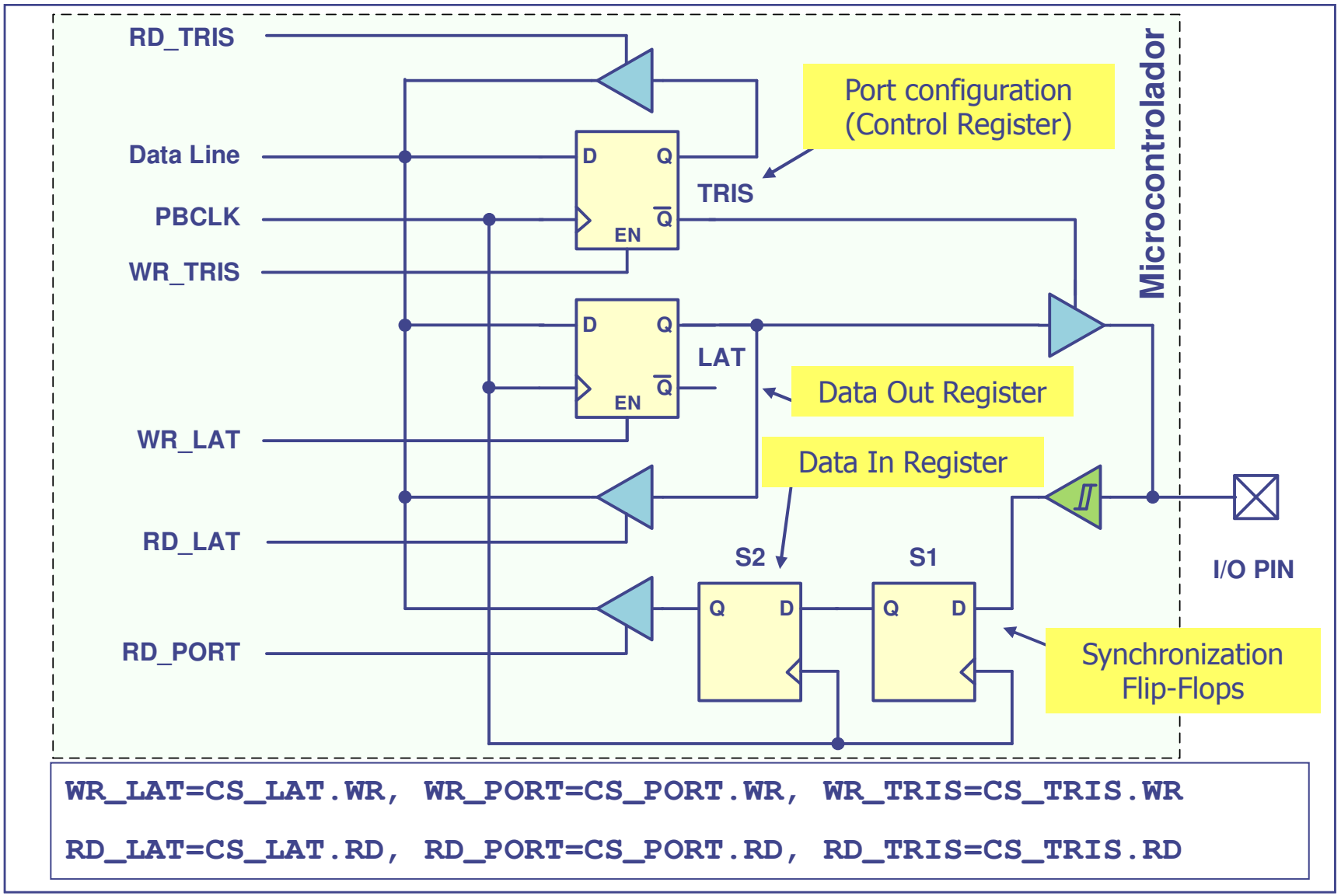
Portos de I/O no PIC32

- Cada um dos portos (B a G) tem associado um total de 12 registos **de 32 bits**. Desses, os que vamos usar são:
  - **TRIS** – usado para configuração do porto (entrada ou saída)
  - **PORT** – usado para ler dados de um porto de entrada
  - **LAT** – usado para escrever dados num porto de saída

- A configuração de cada um dos bits de um porto, como entrada ou como saída, é feita através dos registos **TRIS** ("Tri-state" *registers*)
  - bit **n** do registo TRIS = 1: bit **n** do porto configurado como entrada
  - bit **n** do registo TRIS = 0: bit **n** do porto configurado como saída

- Exemplo para o porto E (8 bits): **`TRISE = 000...10101010`**$_2$
  - portos RE0, RE2, RE4 e RE6 configurados como saída
  - portos RE1, RE3, RE5 e RE7 configurados como entrada

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



Portos de I/O no PIC32

- **Registo TRISx** (TRISB, TRISC, …)
  - registo de configuração (controlo)
  - agrupa todos os flip-flop TRIS dos portos de I/O de 1 bit
  - registos de 32 bits (em que os 16 bits mais significativos são irrelevantes)
- **Registo LATx** (LATB, LATC, …)
  - registo de dados de saída
  - agrupa todos os flip-flops LAT dos portos de I/O de 1 bit
  - registos de 32 bits (em que os 16 bits mais significativos são irrelevantes)

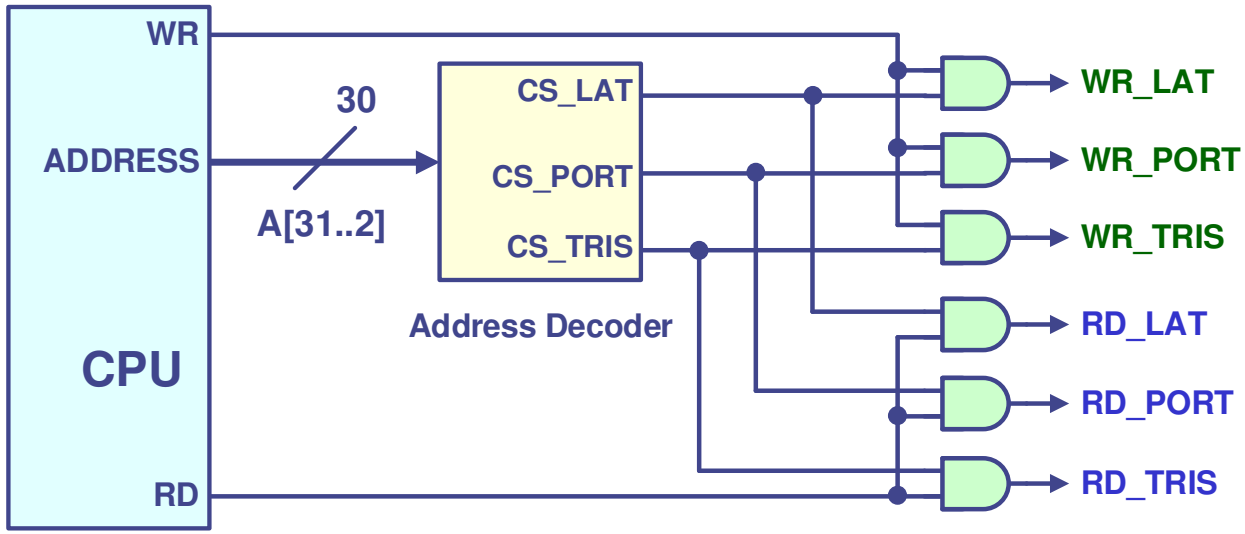
Portos de I/O no PIC32

- **Registo PORTx** (PORTB, PORTC, ...)
  - registo de dados de entrada
  - agrupa todos os flip-flops S2 dos portos de I/O de 1 bit
  - registos de 32 bits (em que os 16 bits mais significativos são irrelevantes)

  - 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* (S1 e S2); 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

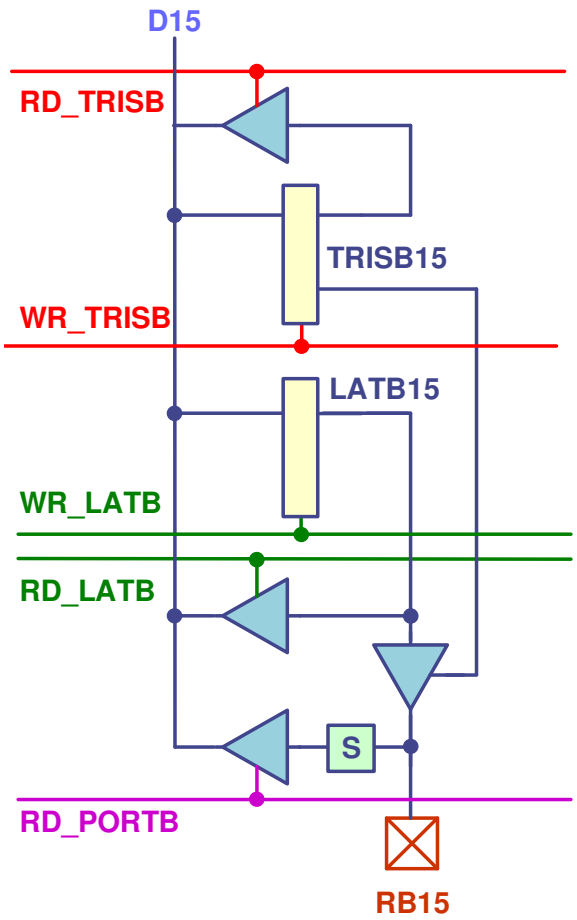
Portos de I/O no PIC32

- Os portos estão mapeados no espaço de endereçamento unificado do PIC32 (ver aula 1), em endereços definidos pelo fabricante
- Os sinais que permitem a escrita e a leitura dos 3 registos de um porto (TRIS, PORT e LAT) são obtidos por descodificação de endereços, em conjunto com os sinais RD e WR gerados pelo CPU

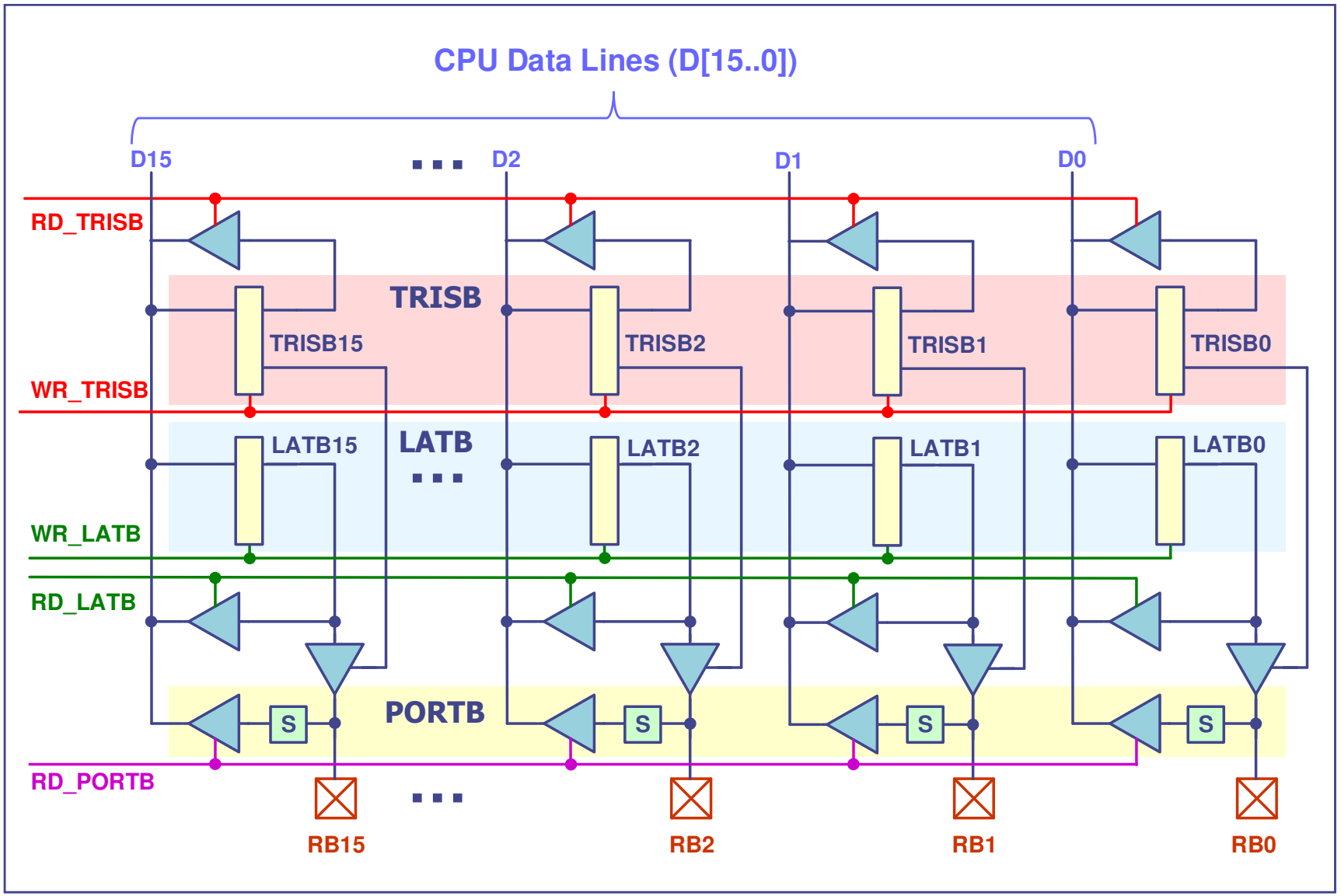


Exemplos de endereços:				
	TRISB:	0xBF886040	TRISE:	0xBF886100
	PORTB:	0xBF886050	PORTE:	0xBF886110
	LATB:	0xBF886060	LATE:	0xBF886120

Modelo simplificado de um porto de I/O no PIC32

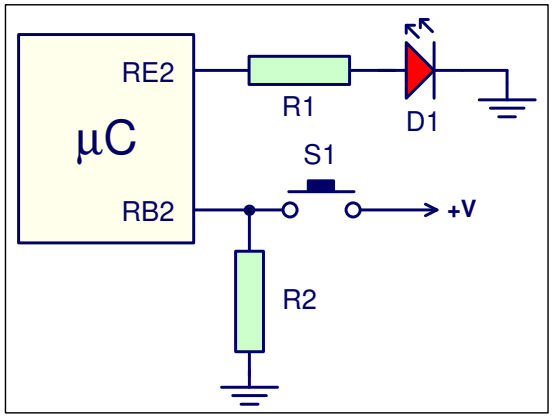


Modelo simplificado de um porto de I/O no PIC32

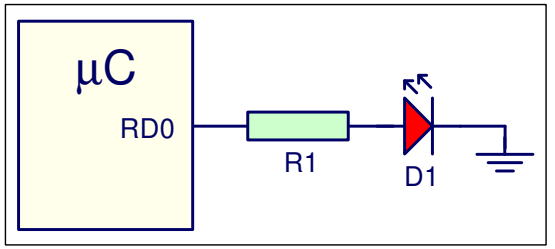


Exemplos de configuração/utilização dos portos

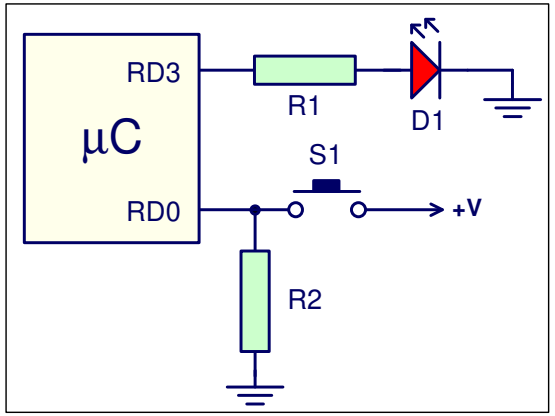
- Exemplo 1:
  - Acender o LED D1 enquanto o *switch* S1 estiver premido; LED ligado ao porto RE2 e *switch* ligado ao porto RB2



- Exemplo 2:
  - Gerar no porto RD0 um sinal de 1 Hz com *duty-cycle* de 10% (i.e. RD0=1 durante 0.1s, RD0=0 durante 0.9s)



- Exemplo 3:
  - Comutar o estado do LED (ligado ao porto RD3) sempre que é detetada uma transição de 0 para 1 no porto RD0



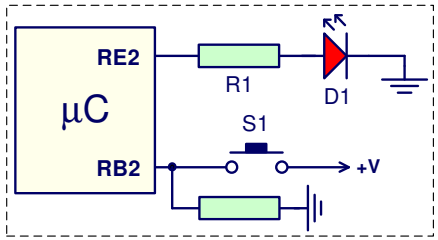
Exemplo de configuração/utilização dos portos

- Definição dos endereços dos portos:

```
.equ ADDR_BASE_HI,0xBF88

.equ TRISB,0x6040    # TRISB address: 0xBF886040
.equ PORTB,0x6050    # PORTB address: 0xBF886050
.equ LATB, 0x6060    # LATB  address: 0xBF886060

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

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

.data

.text
.globl main
```

Exemplo de configuração/utilização dos portos

- **Exemplo 1**: Ler o valor do porto de entrada (RB2) e escrever esse valor no porto de saída (RE2)

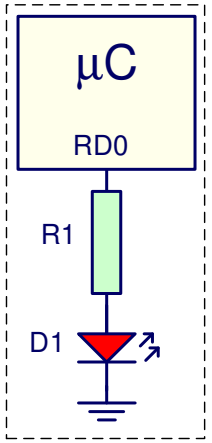



```
        .text
        .globl  main
main:   lui   $t0,ADDR_BASE_HI # $t0=0xBF880000
        lw    $t1,TRISB($t0)   # Address: BF880000 + 00006040
        ori   $t1,$t1,0x0004   # bit2 = 1 (IN)
        sw    $t1,TRISB($t0)   # RB2 configured as IN

        lw    $t1,TRISE($t0)   # Read TRISE register
        andi  $t1,$t1,0xFFFB   # bit2 = 0 (OUT)
        sw    $t1,TRISE($t0)   # RE2 configured as OUT
loop:   lw    $t1,PORTB($t0)   # Read PORTB register
        andi  $t1,$t1,0x0004   # Reset all bits except bit 2

        lw    $t2,LATE($t0)    # Read LATE register
        andi  $t2,$t2,0xFFFB   # Reset bit 2
        or    $t2,$t2,$t1      # Merge data
        sw    $t2,LATE($t0)    # Write LATE register

        j     loop
```

Exemplo de configuração/utilização dos portos

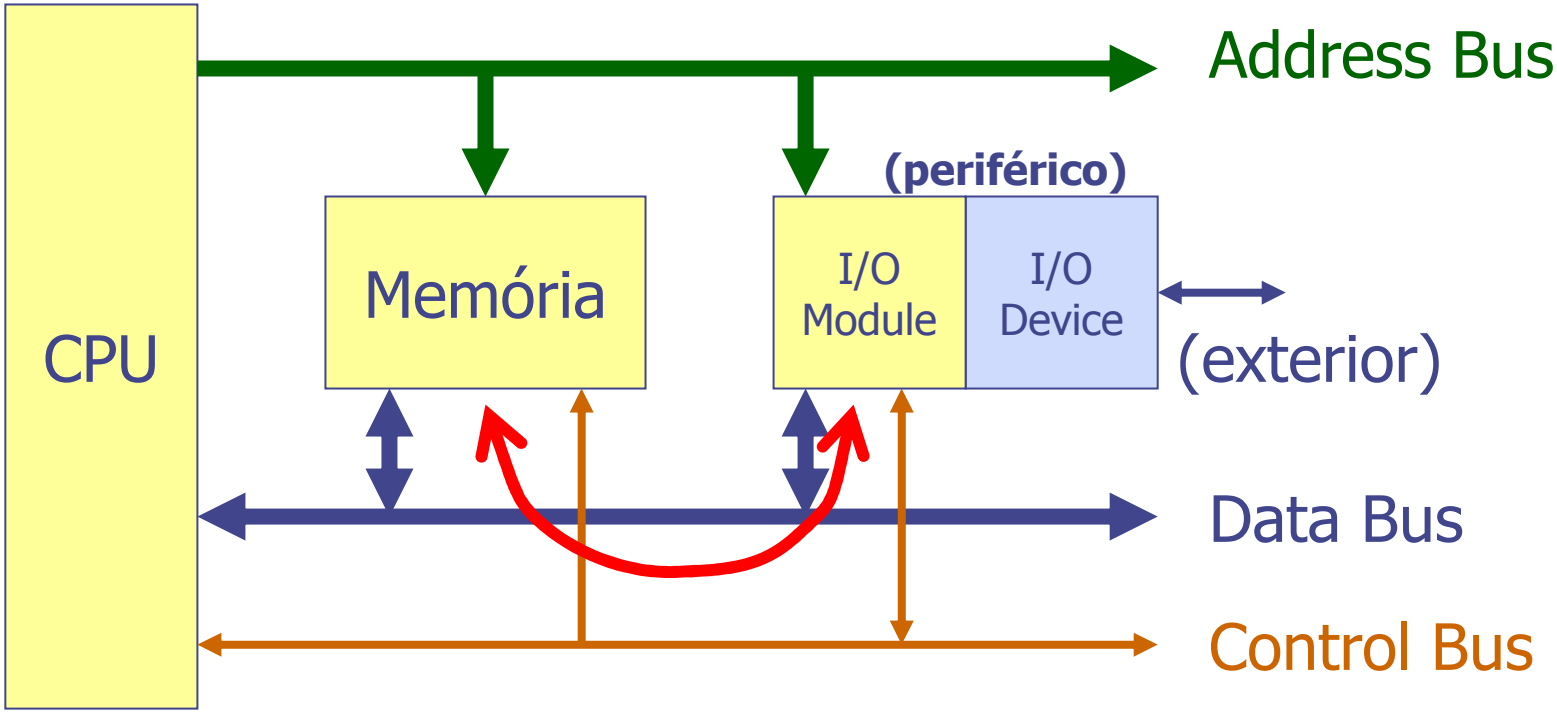
- **Exemplo 2**: gerar no bit 0 do porto D (RD0) um sinal de 1 Hz com *duty-cycle* de 10% (i.e. RD0=1 durante 0.1s, RD0=0 durante 0.9s)

```
        .text
        .globl  main
main:   lui  $t0,ADDR_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
```



- Transferência de dados entre periféricos e memória
  - E/S programada (*polling*)

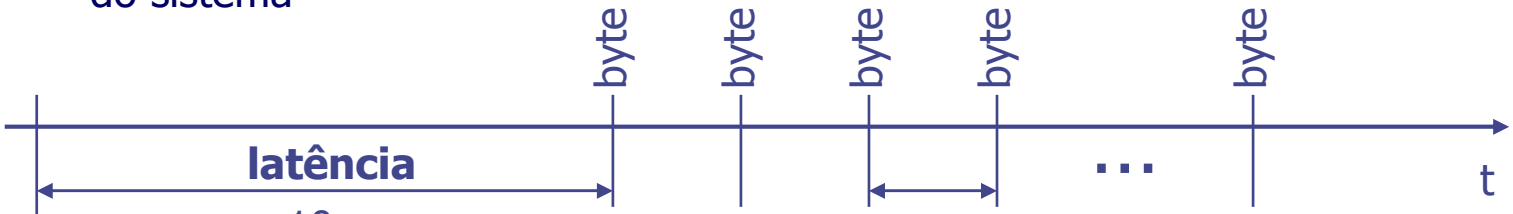
Transferência de informação entre memória e I/O



- CPU – processamento
- Memória – armazenamento (instruções e dados)
- Unidades de I/O – comunicação com dispositivos externos
- Que métodos podem ser usados para gerir a transferência de dados entre os periféricos e a memória?

Transferência de informação entre memória e I/O

- Exemplo: transferência de informação de uma unidade de disco para a memória
  1. **efetuar o pedido** à unidade de disco (por exemplo sector do disco e quantidade de informação pretendida)
  2. **esperar** que a unidade de **disco tenha a informação disponível** na sua memória interna (informação fornecida por 1 bit de um registo de status)
  3. **transferir a informação da memória** da unidade de disco para a memória do sistema



* Estes valores dependem da tecnologia da unidade de disco

- **Latência**: tempo que decorre desde o pedido de informação até à disponibilização do 1º byte de informação
- **Taxa de transferência de pico (*burst*)**: nº máximo de bytes transferidos por segundo, após decorrido o período de latência
- **Taxa de transferência média**: nº total de bytes transferidos / tempo total (incluindo latência)*

Técnicas de gestão de transferência de dados

- E/S programada (***programmed I/O***)
  - método em que o CPU assume o controlo total da transferência de dados entre a memória e um periférico
  - o CPU inicia a operação, verifica repetidamente (*polling*) se o dispositivo está pronto para enviar ou receber dados e depois efetua a transferência
  - *polling* significa que o CPU fica em "espera ativa" (ou seja, ocupado a verificar continuamente o estado do dispositivo), o que pode desperdiçar ciclos de CPU
- E/S por interrupção (***interrupt driven I/O***)
  - método em que o periférico sinaliza o CPU de que está pronto para trocar informação (leitura ou escrita de dados)
  - quando o periférico sinaliza que está pronto, o CPU executa a transferência de dados; enquanto aguarda a interrupção pode continuar a realizar outras tarefas
- E/S por acesso direto à memória (**DMA**)
  - método em que um dispositivo externo ao CPU (DMA) assegura a transferência dos dados diretamente entre a memória e o periférico

E/S Programada

- **Exemplo**: Leitura de N caracteres de um teclado (pseudo-código)

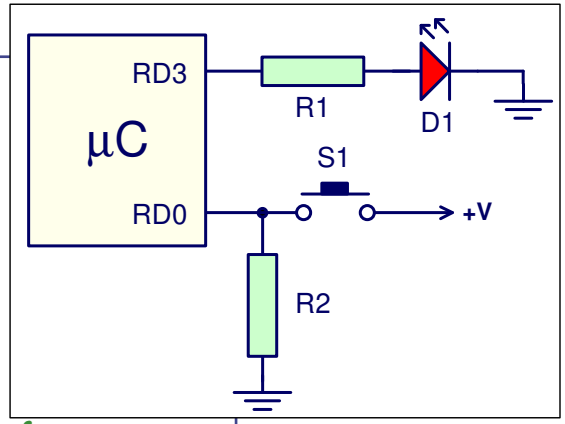
polling

```
nChar = 0
do {
    do {
        Read "Status register" of keyboard I/O Module
    } while ( key not pressed )
    Read character From I/O Module ("data register")
    Write character Into Memory
    nChar = nChar + 1
} while ( nChar < N )
```

- O programa bloqueia no ciclo de verificação de status (*polling*) e só avança quando for premida uma tecla
- Durante esse tempo o CPU não executa qualquer outra ação

E/S Programada (exemplo para o PIC32)

- **Exemplo 3**: comutar o estado do LED (ligado ao porto RD3) sempre que é detetada uma transição de 0 para 1 no porto RD0 (assumindo um sinal isento de *bouncing*).




```
        # config PIC32 ports
        lui   $t0,ADDR_BASE_HI  #
        lw    $t1,TRISD($t0)    #
        ori   $t1,$t1,0x0001    # RD0=1
        andi  $t1,$t1,0xFFF7    # RD3=0
        sw    $t1,TRISD($t0)    #
wh0:    lw    $t1,PORTD($t0)    # while(1) {
        andi  $t2,$t1,0x0001    #
        beq   $t2,$0,wh0        #   while(RD0==0);
        lw    $t3,LATD($t0)     #
        xori  $t3,$t3,0x0008    #
        sw    $t3,LATD($t0)     #   LATD3=!LATD3;
wh1:    lw    $t1,PORTD($t0)    #
        andi  $t1,$t1,0x0001    #
        bne   $t1,$0,wh1        #   while(RD0==1);
        j     wh0               # }
```

polling

polling

E/S programada (resumo)

- O CPU tem que esperar que o periférico esteja disponível para a troca de informação. Essa espera é efetuada num ciclo de verificação da informação de status do periférico, designado por ***POLLING***
- Uma parte substancial do tempo de processamento do CPU pode ser desperdiçado no ciclo de *polling*
- É uma técnica básica, cuja utilização pode ser justificada quando a velocidade do dispositivo periférico não diminui drasticamente a capacidade de processamento do CPU
- O ***overhead*** deste método de transferência (i.e., o número de ciclos de relógio gastos pelo CPU em tarefas que não estão diretamente relacionadas com a transferência de informação – pode ser expressa em %) depende do número de vezes que o ciclo de *polling* for executado
- Uma solução para eliminar o tempo perdido no ciclo de *polling* consiste na utilização da técnica de **E/S por interrupção**

Programação de portos I/O - exercício

S

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)