

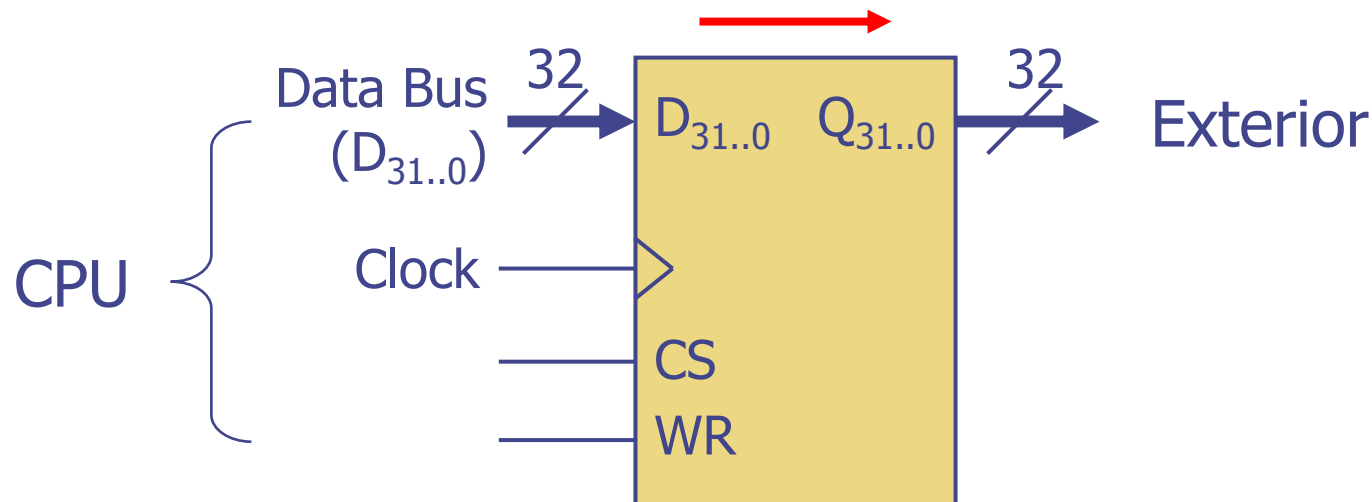
Aula 4

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

José Luís Azevedo, Bernardo Cunha, Tomás Oliveira e Silva

Porto de saída de 32 bits

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

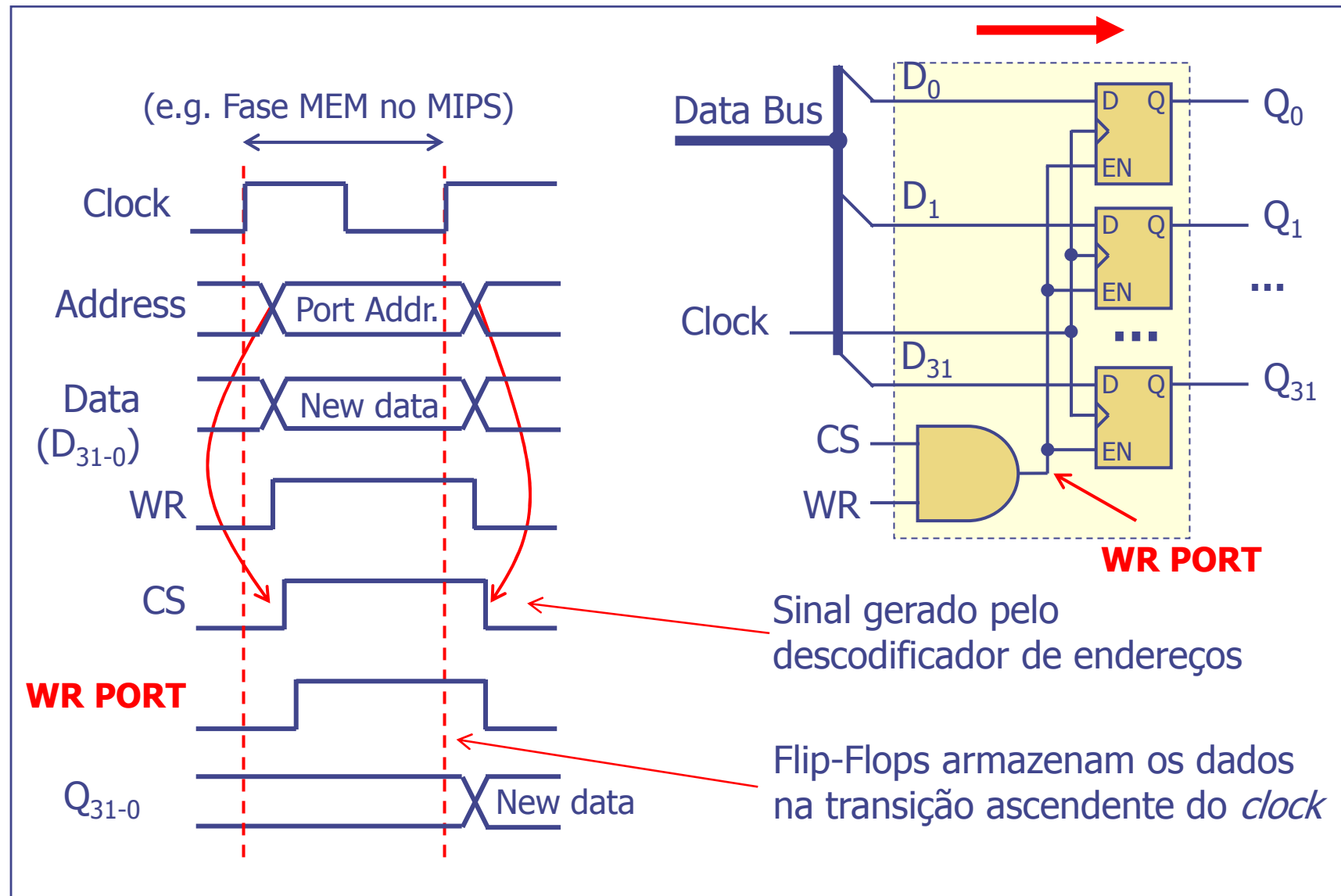


- O porto armazena informação proveniente do CPU, transferida durante uma operação de escrita na memória (estágio MEM nas instruções "**sw**", no caso do MIPS)
- A escrita no porto é feita na transição ativa do relógio se os sinais "**CS**" e "**WR**" estiverem ambos ativos
- O sinal "**CS**" é gerado pelo 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)

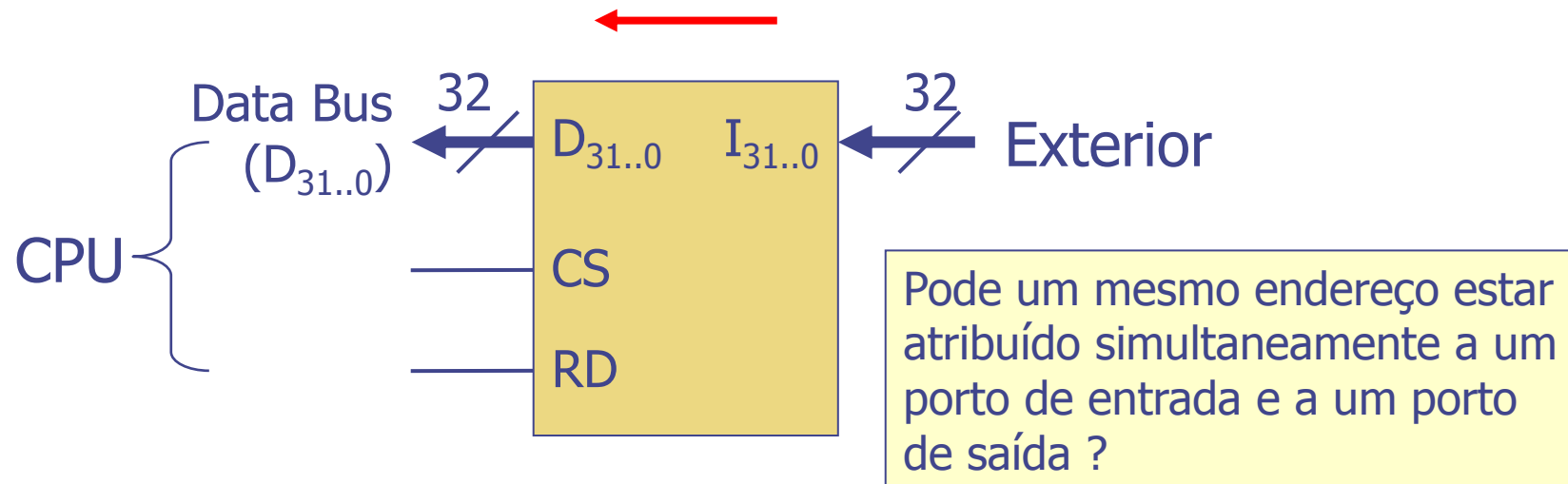
```
entity OutPort is
    port (clk, wr, cs : in  std_logic;
          dataIn      : in  std_logic_vector(31 downto 0);
          dataOut     : out std_logic_vector(31 downto 0));
end OutPort;
architecture behav of OutPort is
begin
    process (clk)
    begin
        if (rising_edge(clk)) then
     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

Porto de saída de 32 bits



Porto de entrada de 32 bits

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

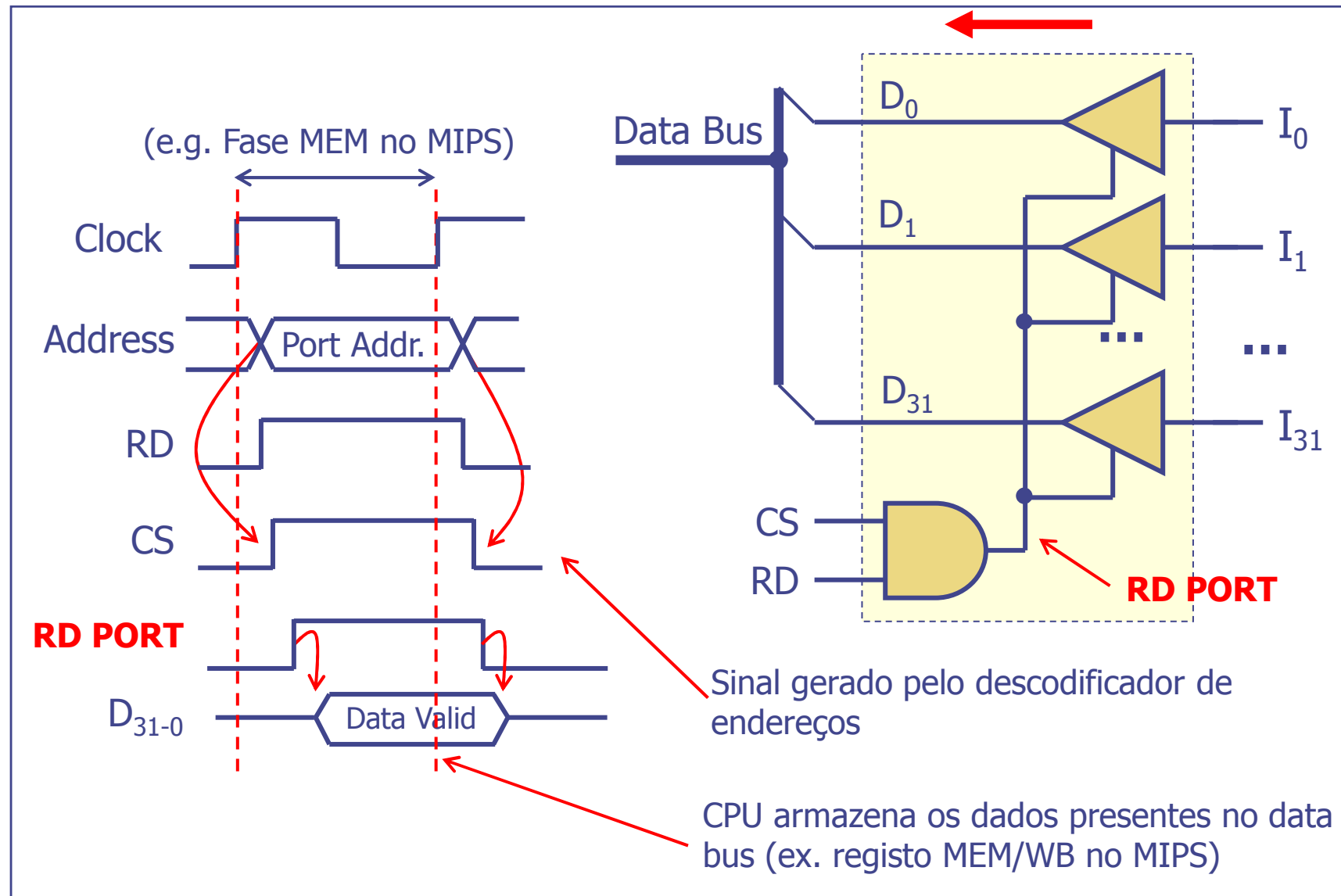


- A informação presente nas 32 linhas de entrada (I_{31..0}) é transferida para o CPU durante uma operação de leitura (estágio MEM nas instruções "**lw**", no caso do MIPS)
- As saídas D_{31..0} têm obrigatoriamente portas *tri-state* que só são ativadas quando estão ativos, simultaneamente, os sinais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

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



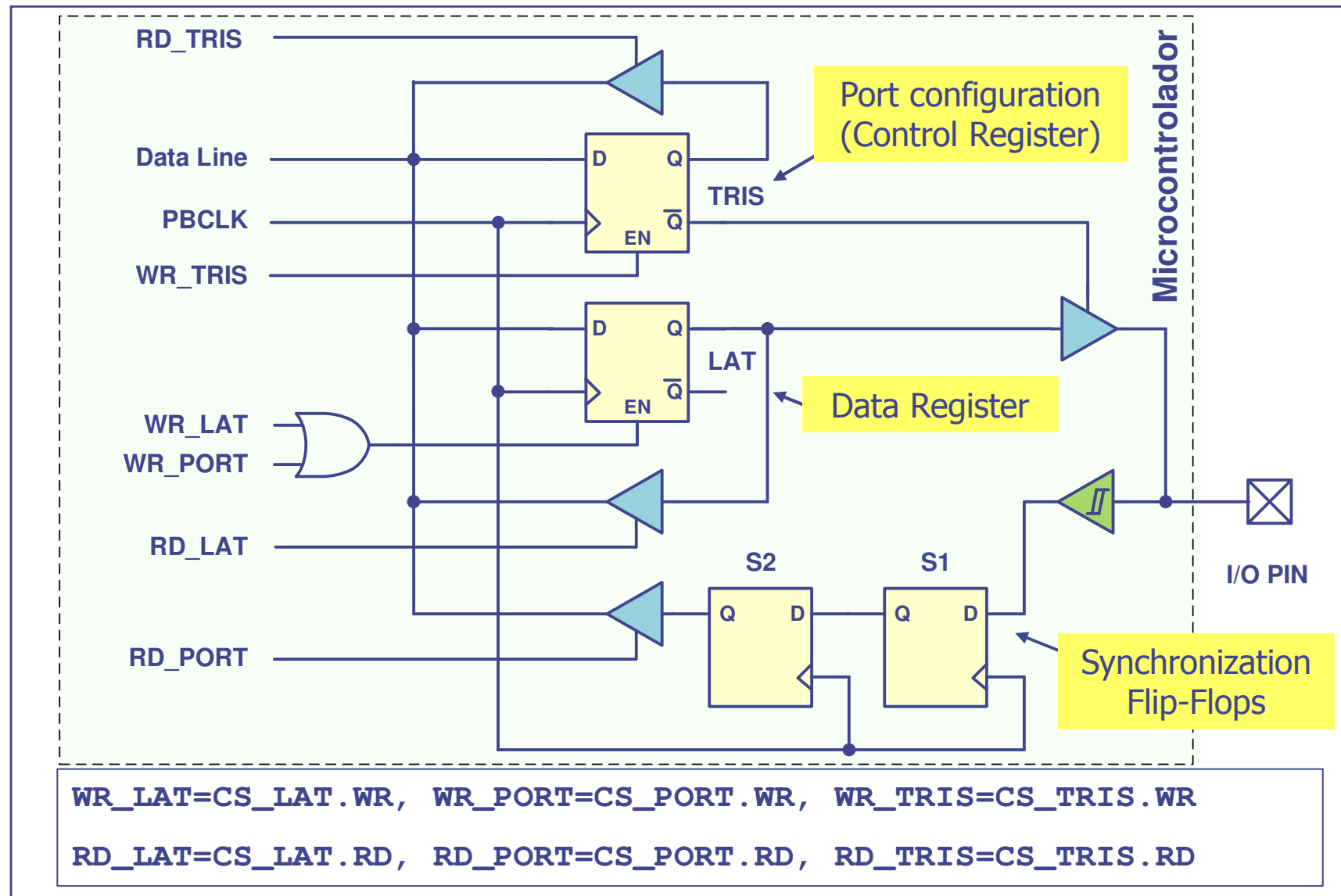
Portos de I/O no PIC32

- O microcontrolador PIC32MX795F512H disponibiliza vários portos de I/O, com várias dimensões (16 bits, no máximo)
 - 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 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 como saída, é feita através dos registros **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** = $000\dots10101010_2$
 - 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

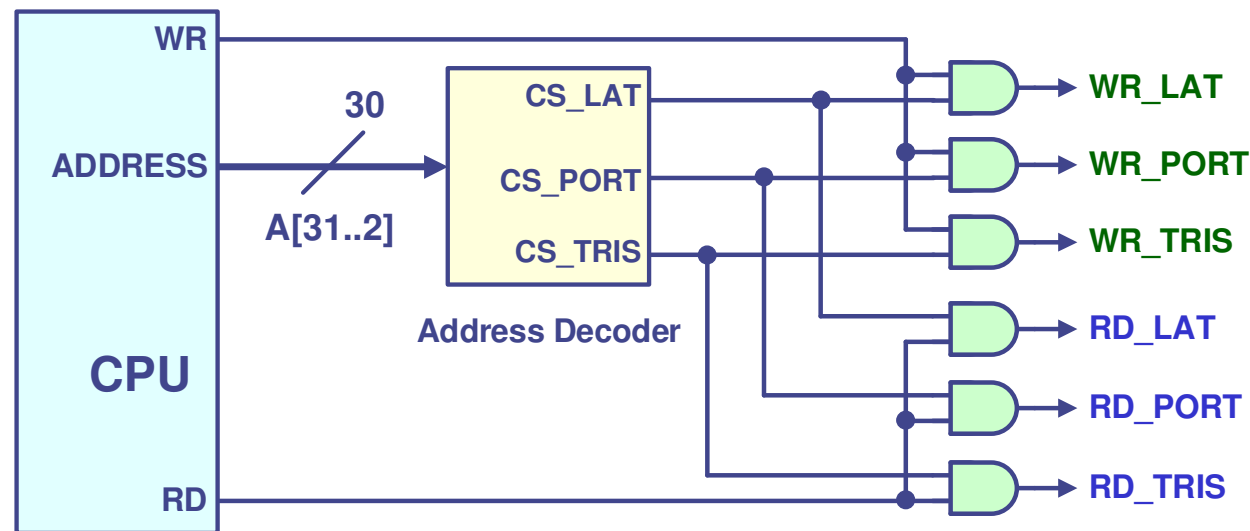


Portos de I/O no PIC32

- Registo TRISx (TRISB, TRISC, ...) agrupa todos os flip-flop TRIS dos portos de I/O de 1 bit; permite a configuração individual de cada um dos bits do porto
- Registo LATx (LATB, LATC, ...) é o registo de dados e agrupa todos os flip-flops LAT dos portos de I/O de 1 bit
- 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

Portos de I/O no PIC32

- A escrita no porto é feita no endereço referenciado pelo identificador **LAT_x**, em que x é a letra que identifica o porto; a leitura do porto é feita do endereço referenciado por **PORT_x**
- 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

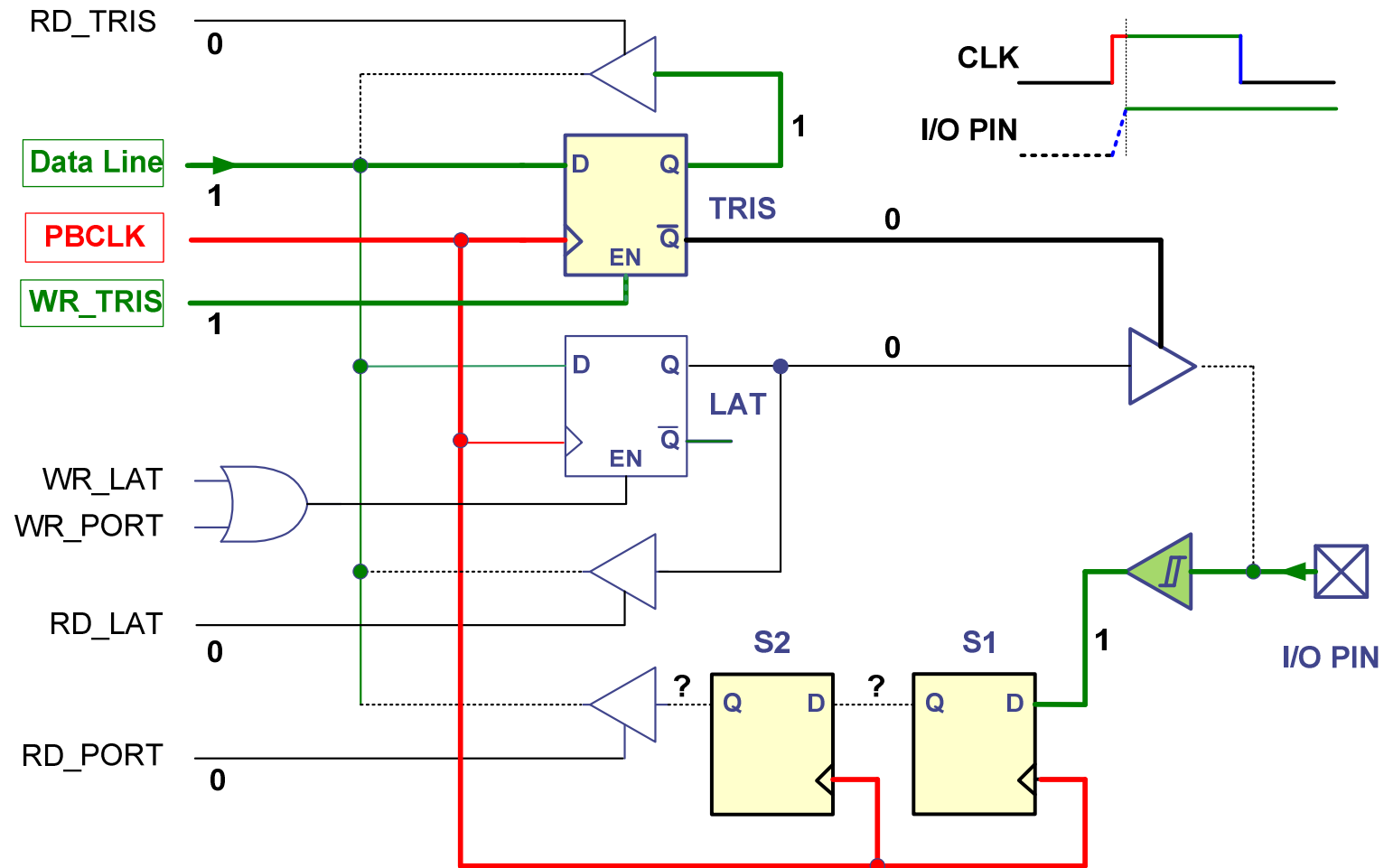


- Exemplos:

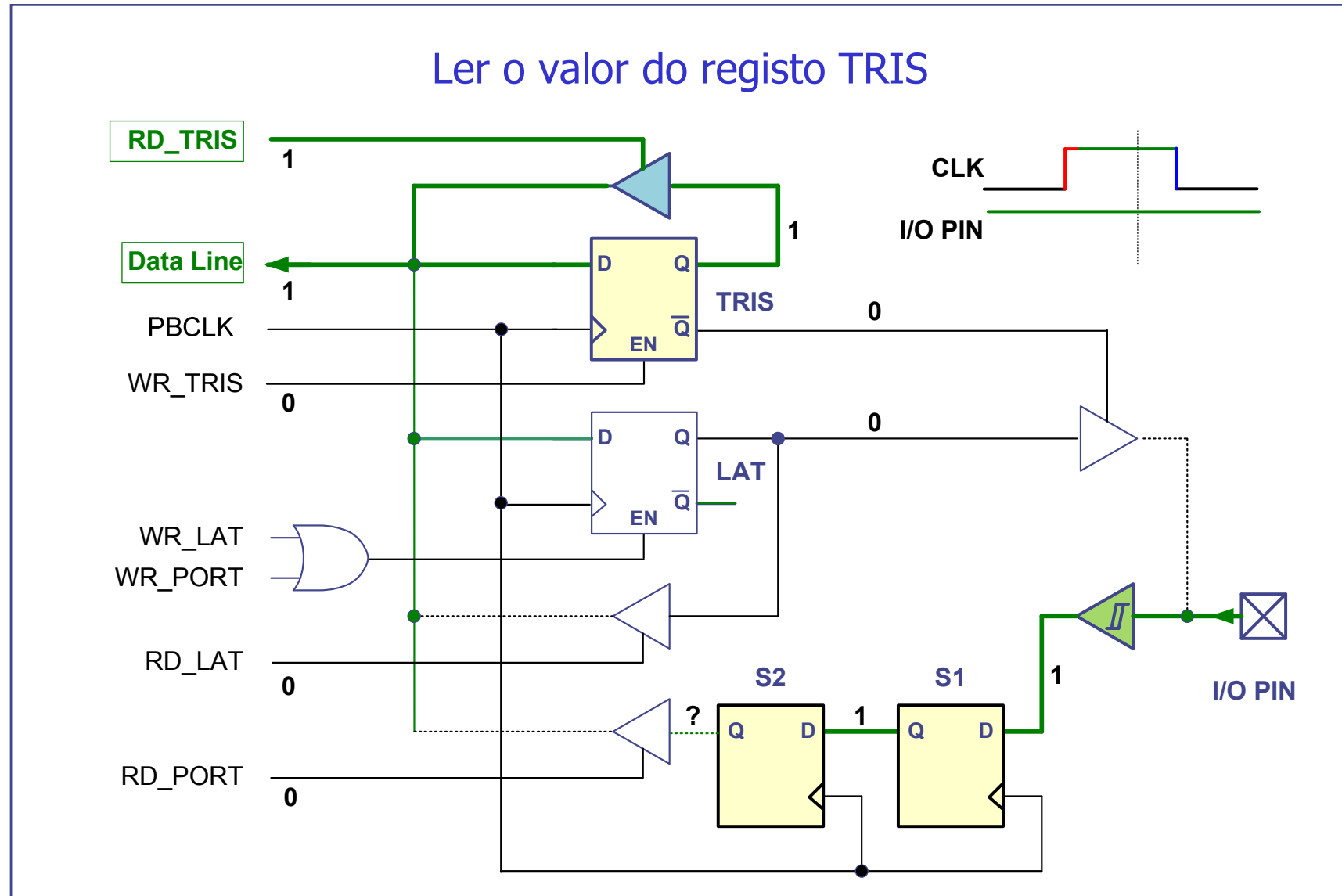
Endereço de TRISB:	0xBF886040
Endereço de PORTB:	0xBF886050
Endereço de LATB:	0xBF886060

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

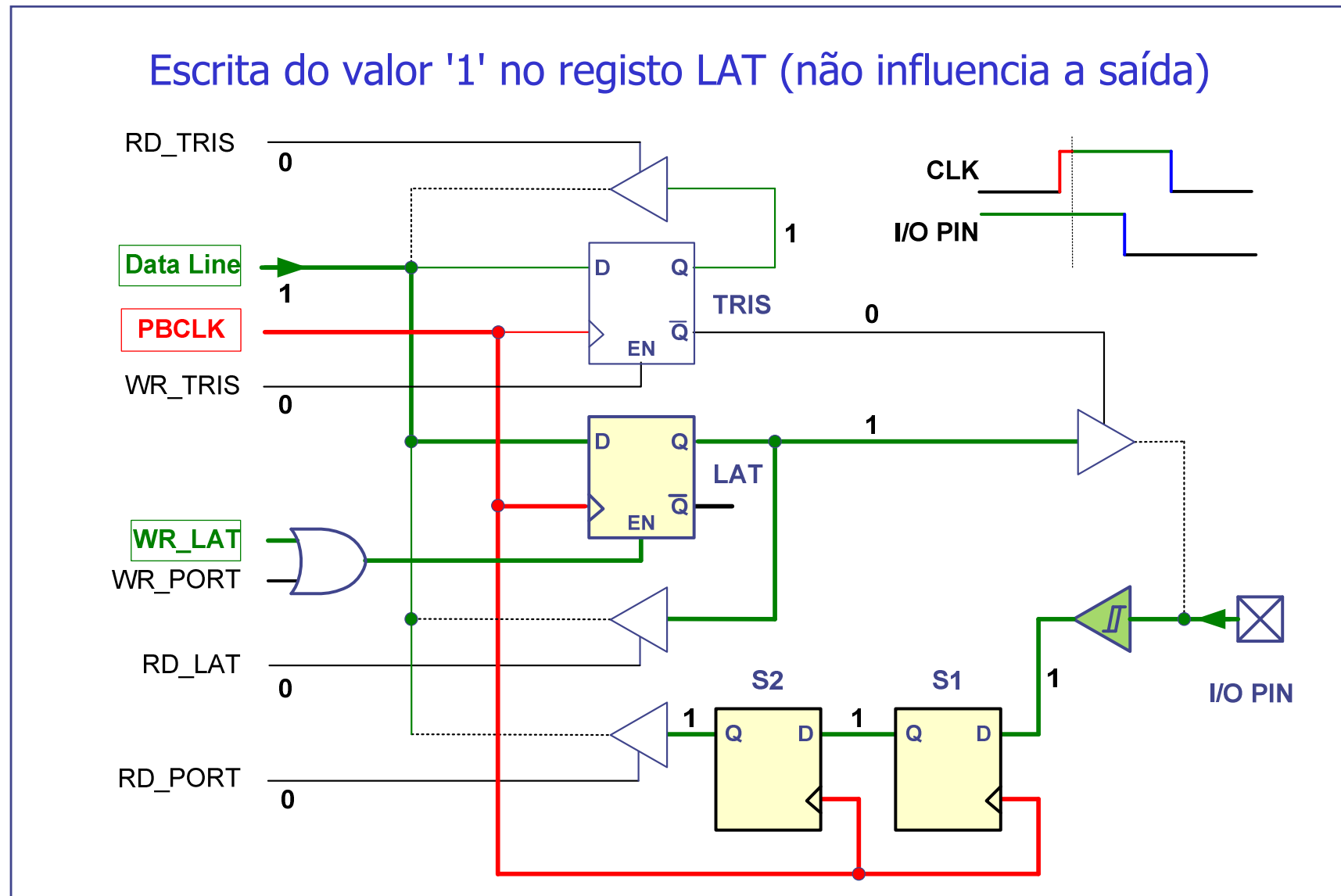
Definir pino como porto de entrada – Escrita do valor '1' no TRIS



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



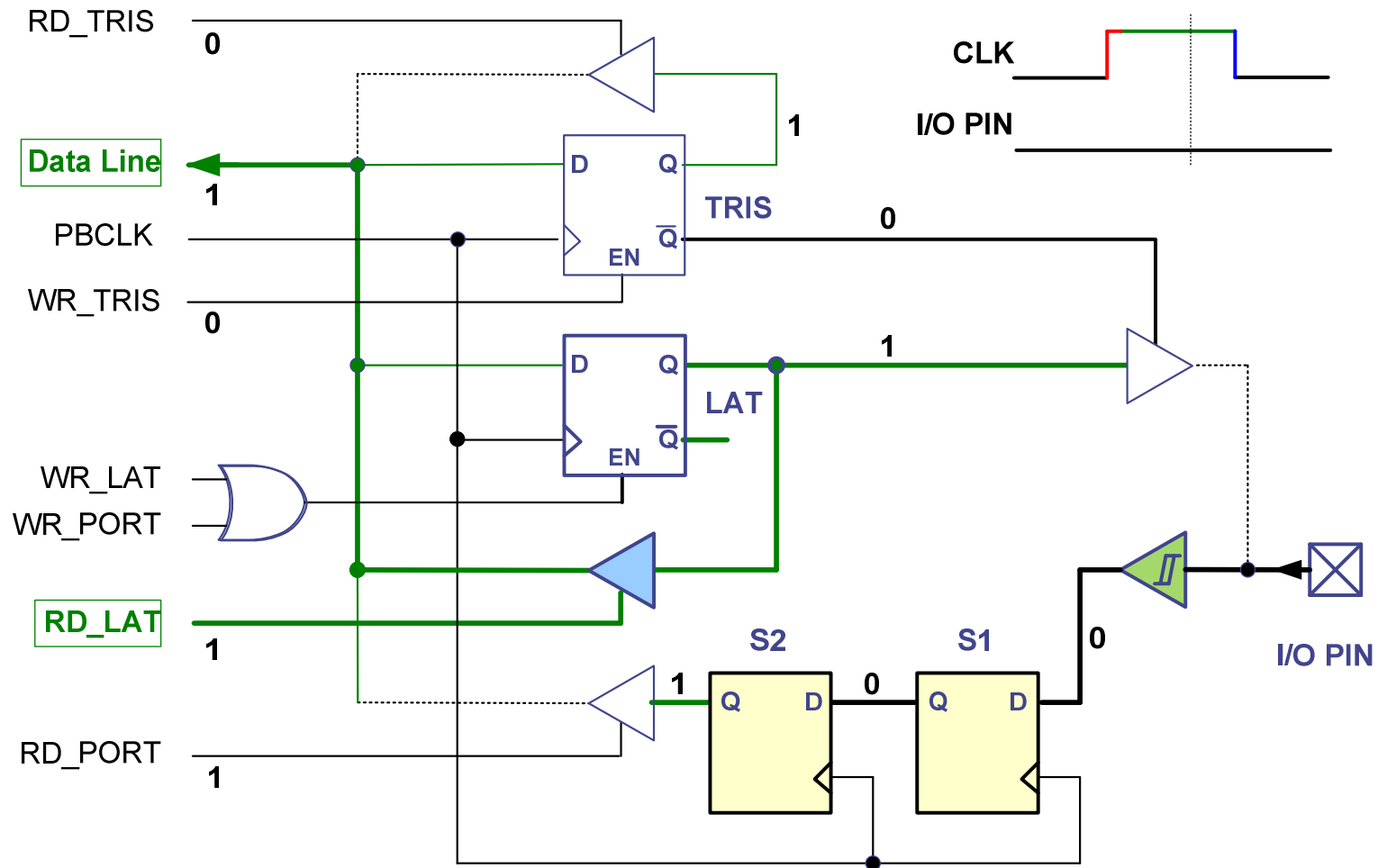
Leitura do Porto de entrada (com 2 ciclos de relógio de atraso)

The diagram illustrates the internal logic for reading an input port. It features several control signals and a data path:

- Control Signals:** RD_TRIS (0), WR_TRIS (0), WR_LAT (0), RD_LAT (0), and RD_PORT (1).
- Data Line:** A green line representing the data being read, currently showing a value of 1.
- Flip-Flops:** Two sequential flip-flops, S2 and S1, are used to delay the data by two clock cycles. S2's output (Q) is 1, and S1's output (Q) is 0.
- Logic Elements:** The circuit includes inverters, a 2-input AND gate (for WR_LAT and WR_PORT), and a 3-input OR gate (for RD_TRIS, WR_TRIS, and RD_PORT).
- Timing Diagram:** Shows the CLK signal (a square wave) and the I/O PIN signal (a square wave that is high during the read operation).

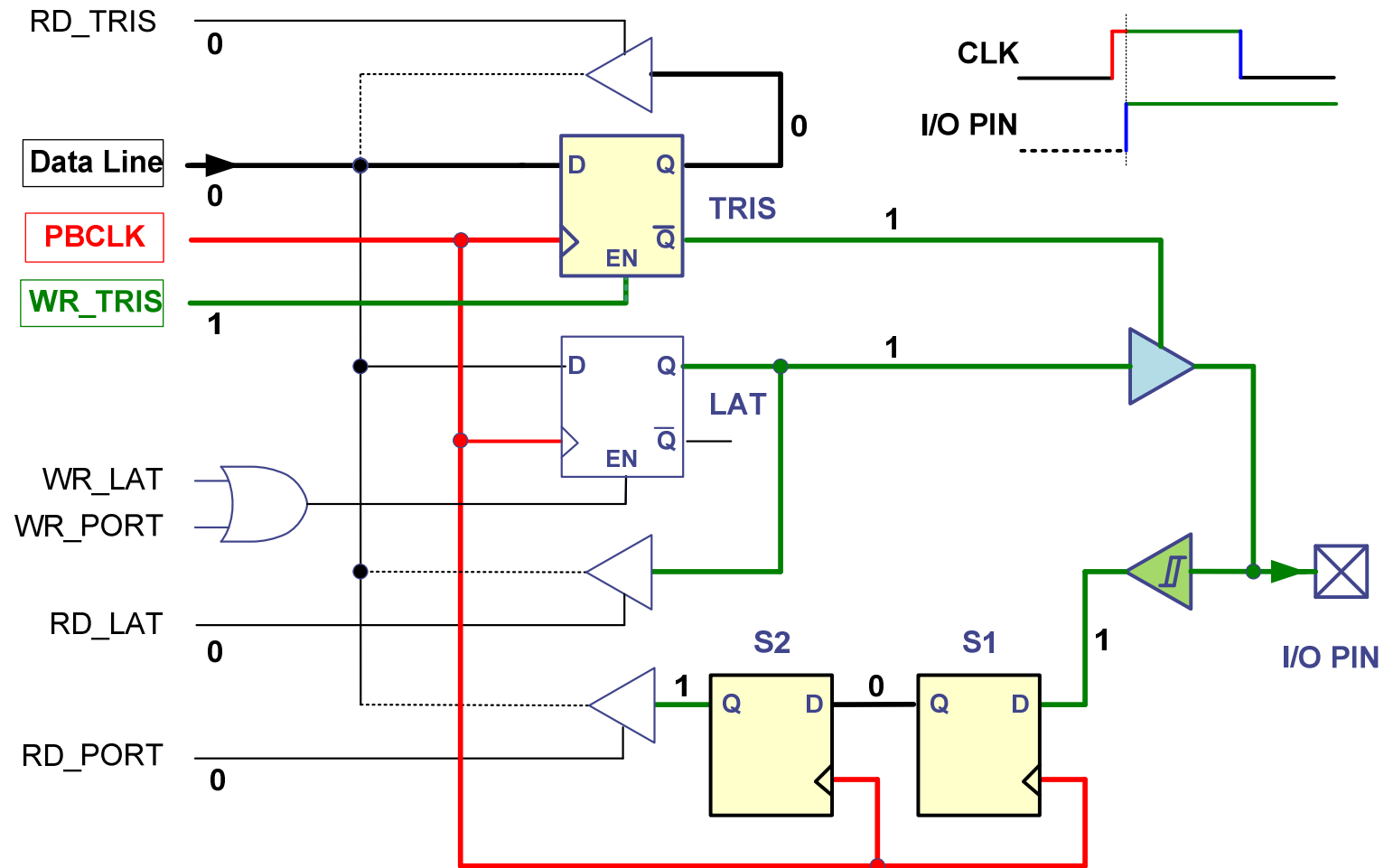
Modelo simplificado de um porto de I/O de 1 bit no PIC32

Leitura do valor do registro LAT (pino é uma entrada)



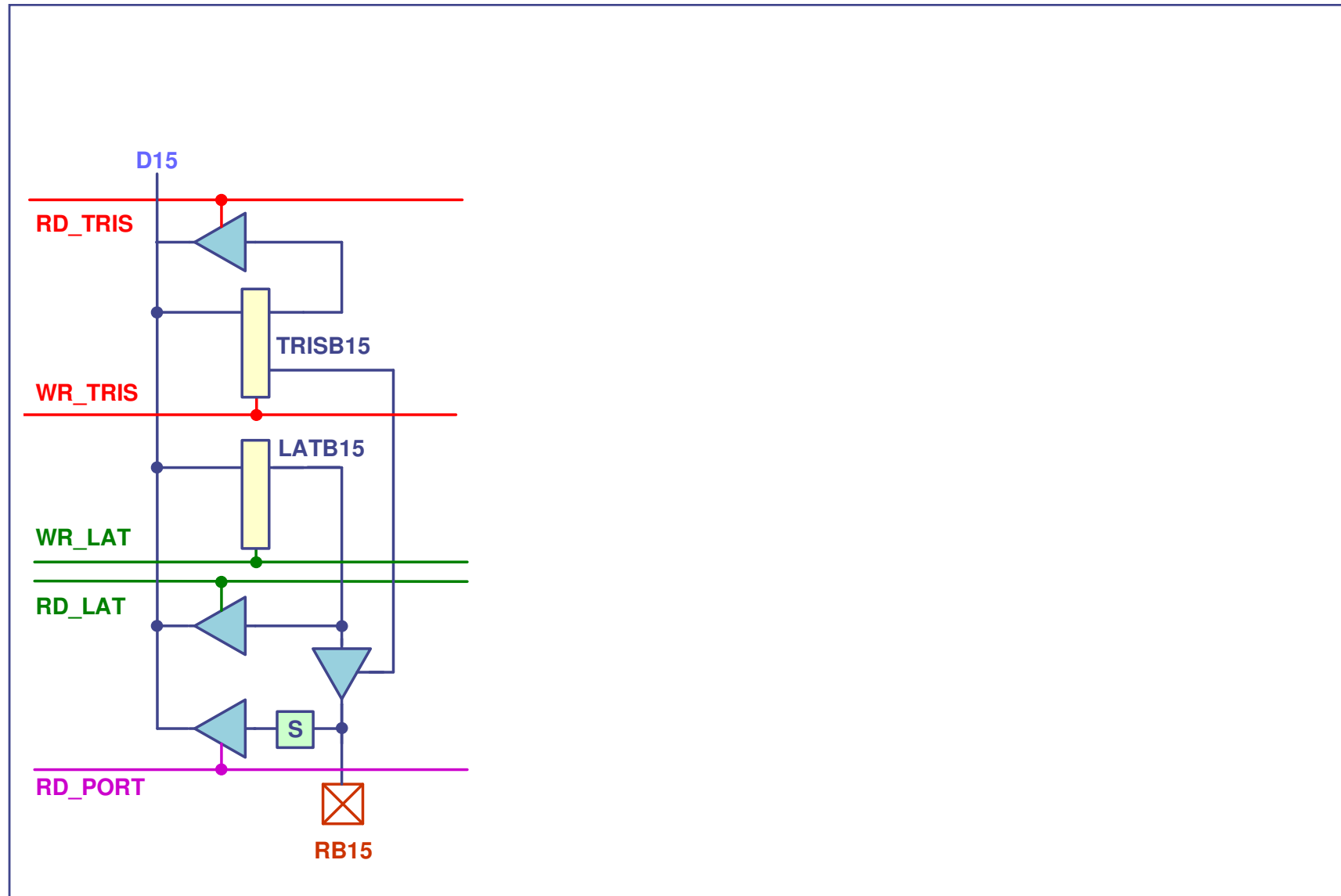
Modelo simplificado de um porto de I/O de 1 bit no PIC32

Definir pino como porto de saída – Escrita do valor '0' no TRIS

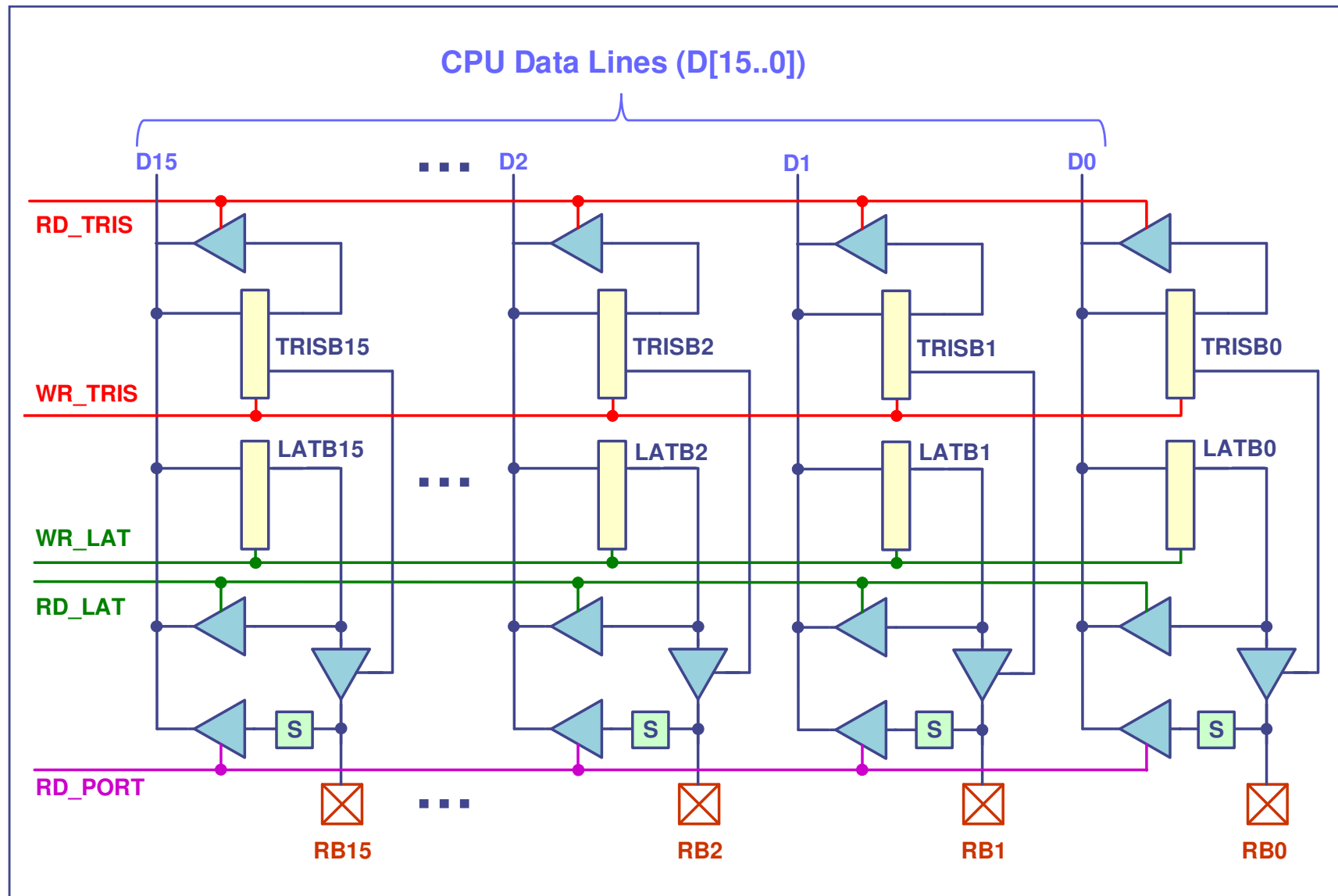




Modelo simplificado de um porto de I/O no PIC32



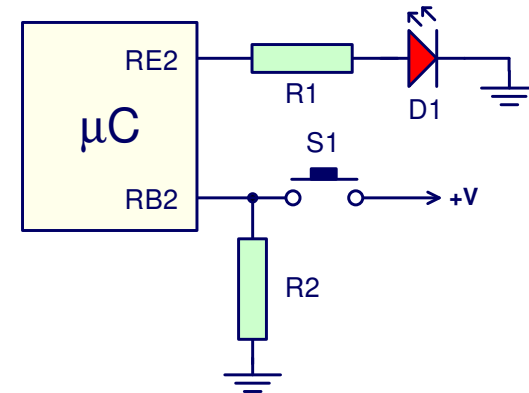
Modelo simplificado de um porto de I/O no PIC32



Exemplo de configuração/utilização dos portos

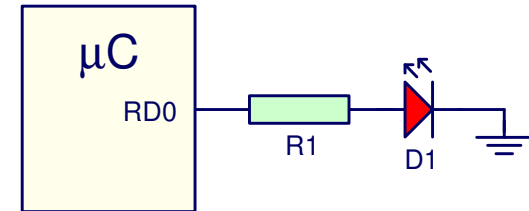
- Exemplo 1:

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



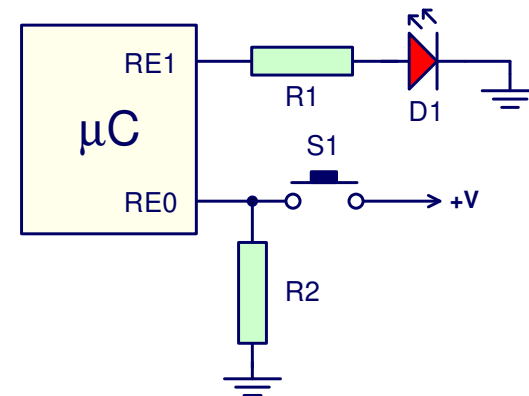
- Exemplo 2:

- 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:

- Manter o LED D1 apagado enquanto o *switch* S1 estiver premido, e aceso na situação contrária; LED D1 ligado ao porto RE1 e *switch* ligado ao porto RE0



Exemplo de configuração/utilização dos portos

- Definição dos endereços dos portos:

```
.equ  SFR_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

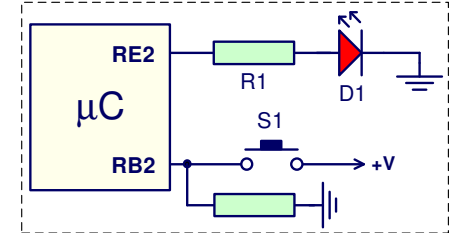
.text

.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, SFR_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, 0xFFFFB   # 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, 0xFFFFB   # 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, 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)

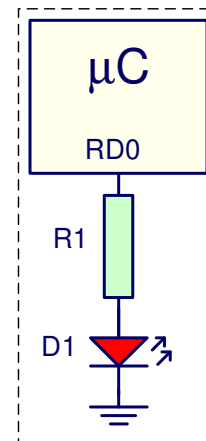
loop: lw   $t1, LATD($t0)    # Read LATD
      ori  $t1, $t1, 0x0001  # Modify bit 0 (set)
      sw   $t1, LATD($t0)    # Write LATD

# 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

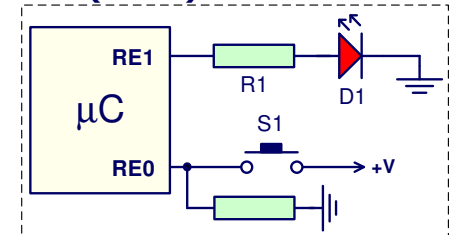
# wait 900 ms (e.g., using MIPS core timer)

      j    loop
```



Exemplo de configuração/utilização dos portos

- Exemplo 3: em ciclo infinito, ler o valor do porto de entrada (RE0) e escrever esse valor, negado, no porto de saída (RE1)



```
.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, 0xFFFD   # bit1 = 0 (OUT)
      sw   $t1, TRISE($t0)    # TRISE configured

loop: lw   $t1, PORTE($t0)    # Read PORTE register
      andi $t1, $t1, 0x0001   # Reset all bits except bit 0
      xori $t1, $t1, 0x0001   # Negate bit 0
      sll  $t1, $t1, 1        #
      lw   $t2, LATE($t0)     # Read LATE register
      andi $t2, $t2, 0xFFFD   # Reset bit 1
      or   $t2, $t2, $t1      # Merge data
      sw   $t2, LATE($t0)     # Write LATE register
      j    loop
```

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)