

# Interface entre Processador e Periféricos

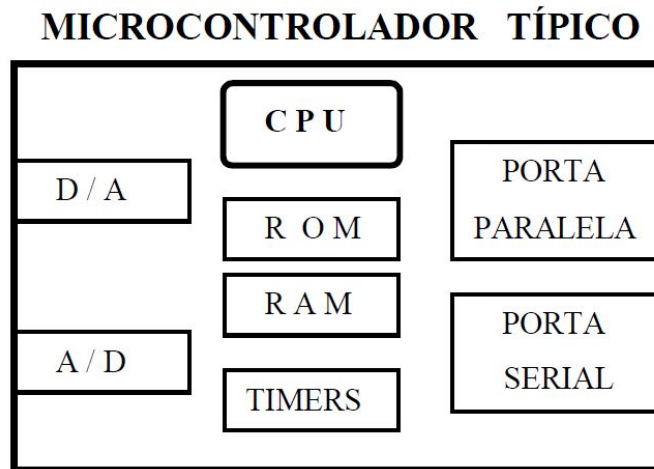
---

Sub-sistema de entrada e saída –  
E/S (I/O)

# Interface entre Processador e Periféricos

---

- Periféricos do processador
  - Circuitos na periferia do processador (CPU)



- A comunicação entre o processador e os periféricos é feita através de **registradores**
  - Os registradores podem estar dentro dos periféricos ou entre o processador e os periféricos
  - Podem ser de leitura, de escrita ou ambos
  - Esses registradores também são chamados de portas de E/S

# Interface entre Processador e Periféricos

---

- Da mesma maneira que o processador acessa a memória fornecendo endereços, outros periféricos também tem endereços de acesso, os quais correspondem aos registradores
- O endereçamento dos registradores pode seguir duas abordagens
  - Mapeamento em memória
  - Mapeamento em portas de E/S

# Interface entre Processador e Periféricos

---

- Mapeamento em memória
    - Espaço de endereçamento único
      - Parte usado para acessar a memória
      - Parte usado para acessar periféricos
      - Divisão do espaço definida pelo projetista
      - Exemplo: espaço de endereçamento 0 – 255 (8 bits)
        - 0 – 249: memória (250 endereços de memória)
        - 250 – 255: periféricos (6 endereços de periféricos)
    - Periféricos acessados utilizando instruções de acesso à memória
      - Load/Store
    - Exemplos
      - MIPS
      - Processadores da Motorola
-

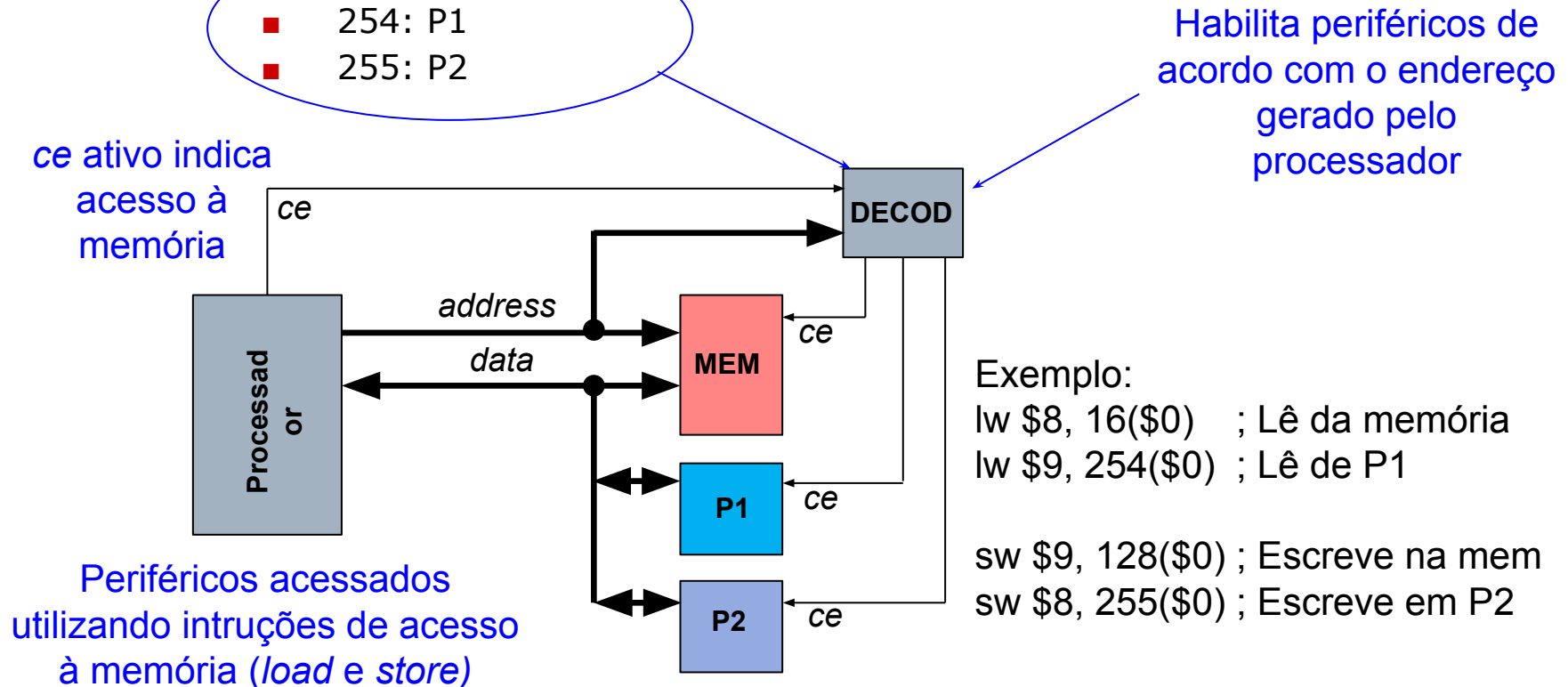
# Interface entre Processador e Periféricos

## □ Mapeamento em memória

### ■ Exemplo: barramento de endereços de 8 bits

#### □ Espaço de endereçamento: 0 – 255

- 0 – 253: memória
- 254: P1
- 255: P2



# Interface entre Processador e Periféricos

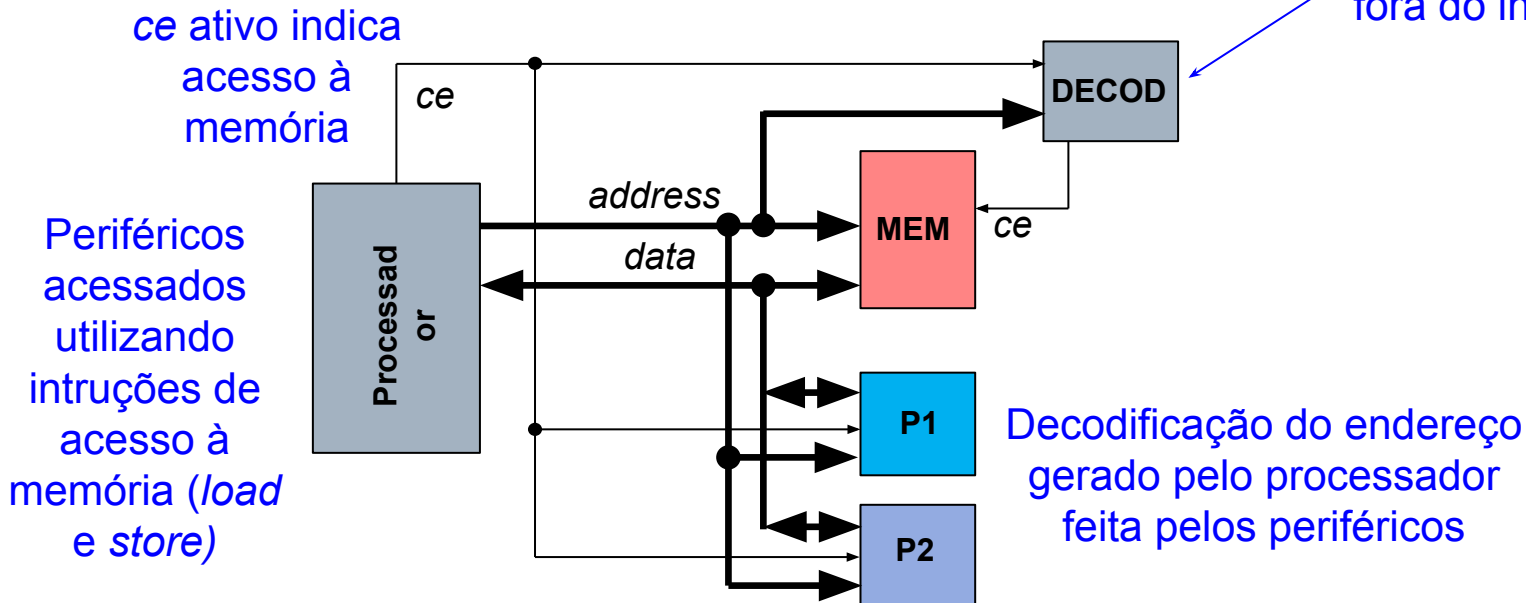
## □ Mapeamento em memória

### ■ Exemplo: barramento de endereços de 8 bits

#### □ Espaço de endereçamento: 0 – 255

- 0 – 253: memória
- 254: P1
- 255: P2

Desabilita a memória quando o endereço gerado pelo processador estiver fora do intervalo 0 - 253



# Interface entre Processador e Periféricos

## □ Mapeamento em memória

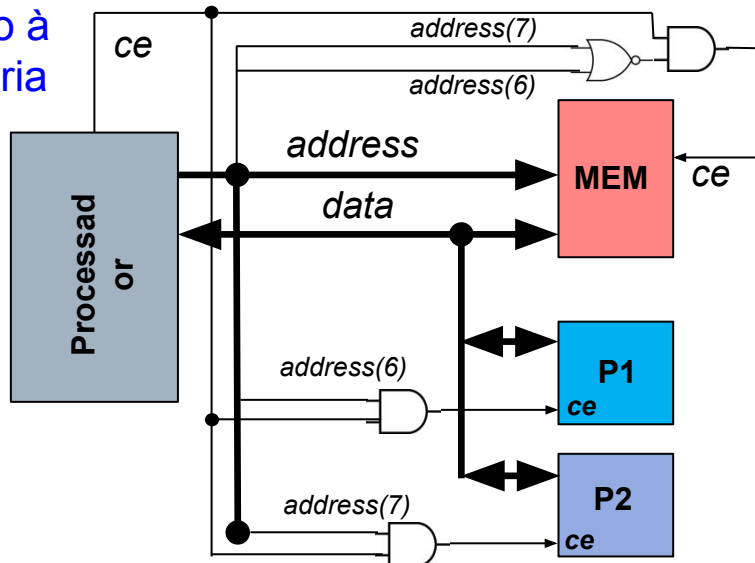
### ■ Exemplo: barramento de endereços de 8 bits

#### □ Espaço de endereçamento: 0 – 255

- 0 – 63 (**00**000000-**00**111111): memória
- 64 – 127 (**01**000000-**01**111111): P1
- 128-191 (**10**000000-**10**111111): P2
- 192-255 (**11**000000-**11**111111): P1 e P2 (simultâneos)

ce ativo indica  
acesso à  
memória

Periféricos  
acessados  
utilizando  
instruções de  
acesso à  
memória (*load*  
e *store*)



Elimina decodificadores, os quais  
podem ser bem grandes em  
termos de portas lógicas

# Interface entre Processador e Periféricos

---

- Mapeamento em portas de E/S
    - Dois espaços de endereçamento isolados
      - Um espaço é usado para acessar a memória
      - Outro é usado para acessar as interfaces de E/S
      - Exemplo considerando um único barramento de endereços de 8 bits
        - 0 – 255: memória (256 endereços de memória)
        - 0 – 255: periféricos (256 endereços de periféricos)
      - O processador tem uma saída (e.g. io) que indica o espaço de endereçamento em uso (e.g. io = 1: periférico)
    - Periféricos são acessados utilizando instruções específicas
      - In/Out
    - Exemplos
      - Processadores da Intel
-



# Interface entre Processador e Periféricos

## □ Mapeamento em portas de E/S

### ■ Exemplo: barramento de endereços de 8 bits

#### □ Espaço de endereçamento: 0 – 255

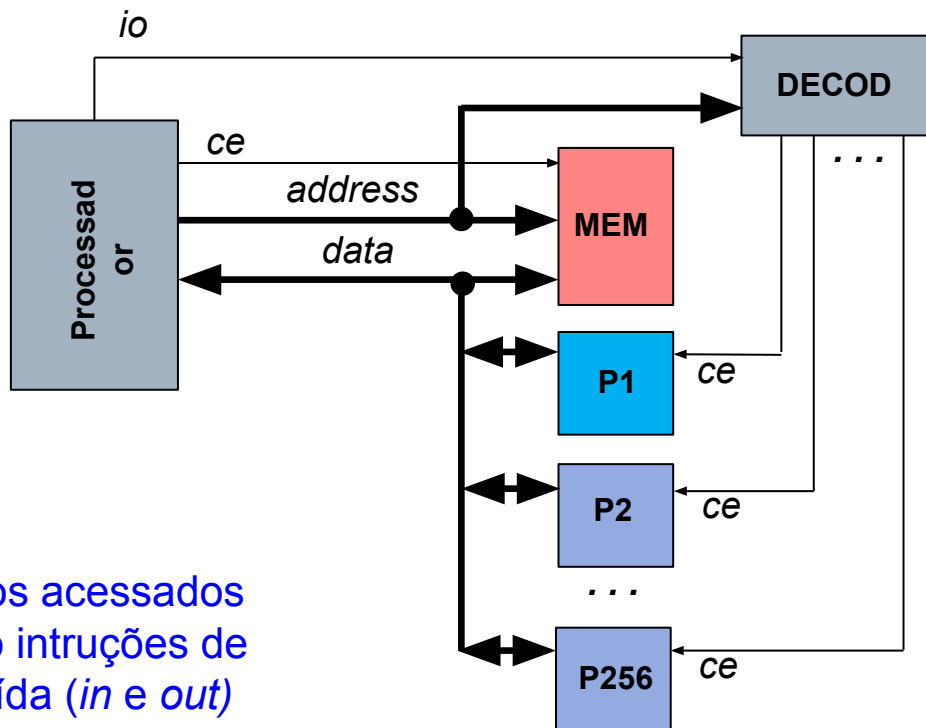
■ 0 – 255: memória

■ 0 – 255: periféricos (0: P1; 1:P2; ... ;255: P256)

Habilita periféricos de acordo com o endereço gerado pelo processador

io ativo indica acesso a periféricos

ce ativo indica acesso à memória



Exemplo:

`lw $8, 0($0)` ; Lê da memória

`in $9, 0($0)` ; Lê de P1 (io=1)

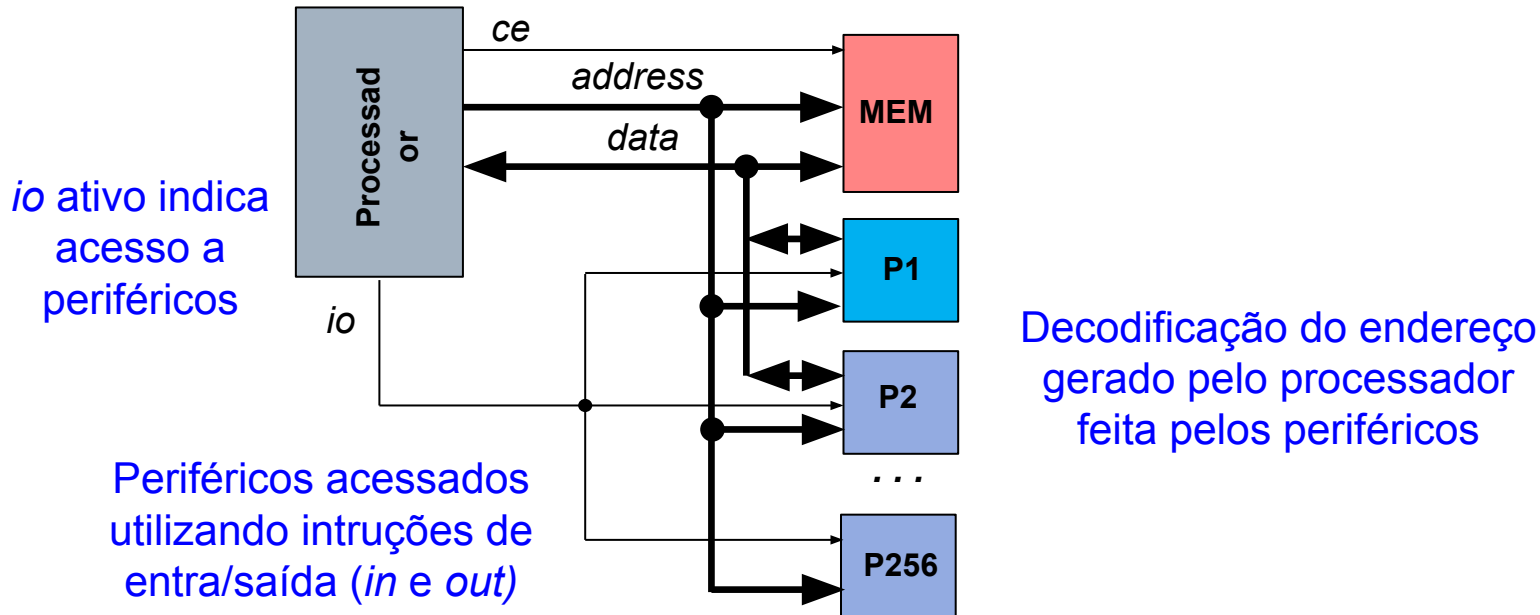
`sw $9, 1($0)` ; Escreve na mem

`out $8, 1($0)` ; Escr. em P2 (io=1)

Periféricos acessados utilizando instruções de entra/saída (*in* e *out*)

# Interface entre Processador e Periféricos

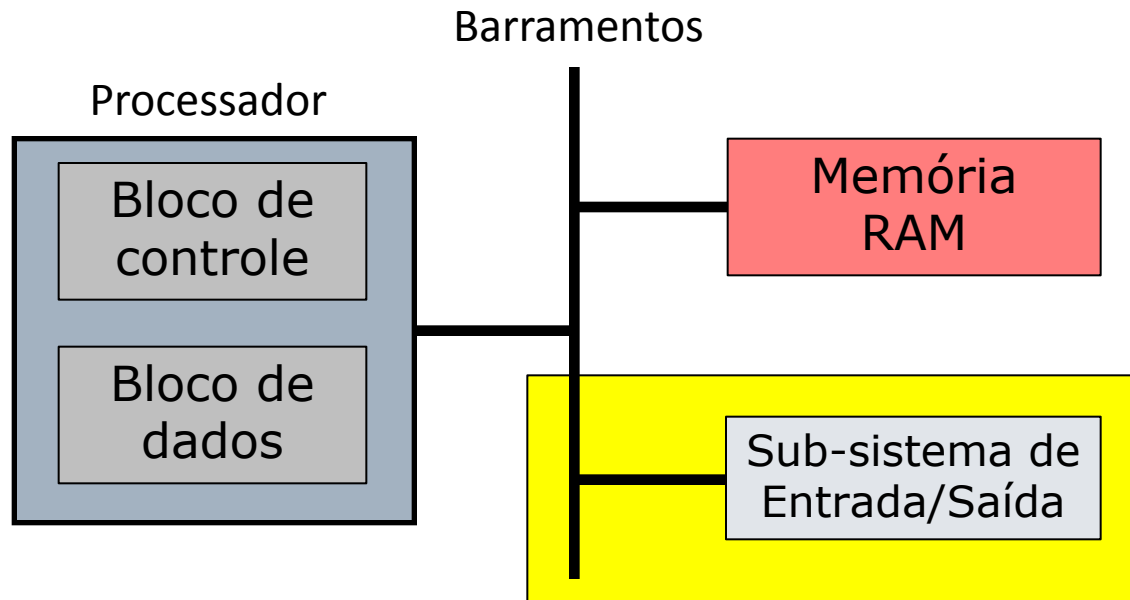
- Mapeamento em portas de E/S
  - Exemplo: barramento de endereços de 8 bits
    - Espaço de endereçamento: 0 – 255
      - 0 – 255: memória
      - 0 – 255: periféricos (0: P1; 1:P2; ... ;255: P256)



# Interface entre Processador e Periféricos

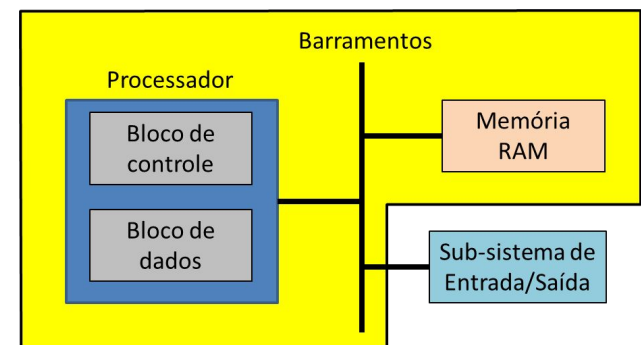
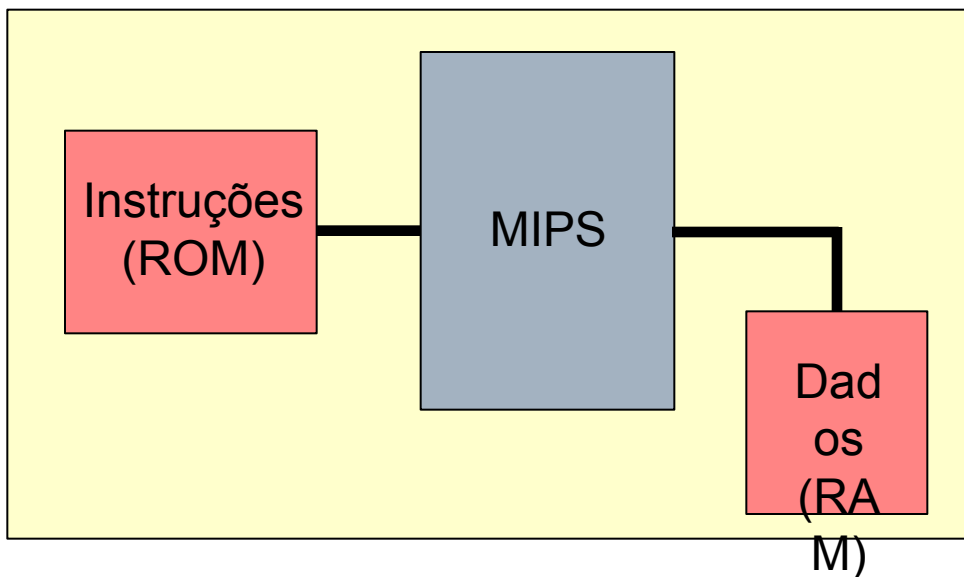
---

- Estrutura básica de um sistema computacional
  - Principais componentes



# Interface entre Processador e Periféricos

- Estrutura básica de um sistema computacional
  - Sistema atual

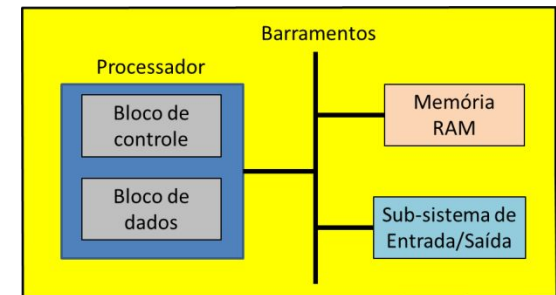
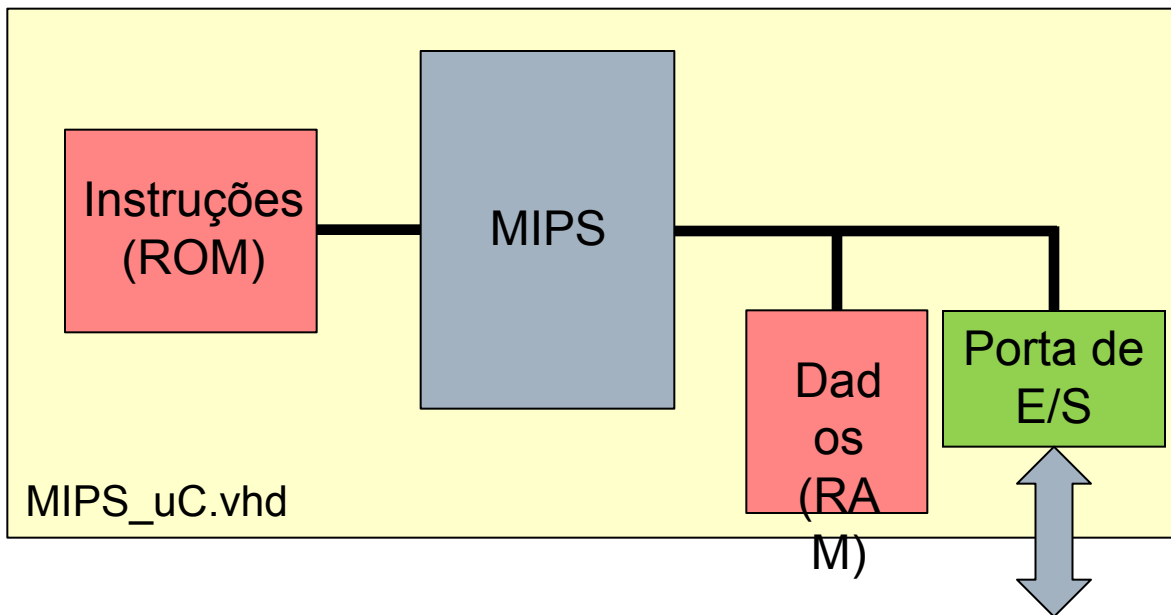


O objetivo do trabalho é adicionar ao sistema uma porta de E/S

# Interface entre Processador e Periféricos

- Estrutura básica de um sistema computacional
  - Porta de E/S adicionada

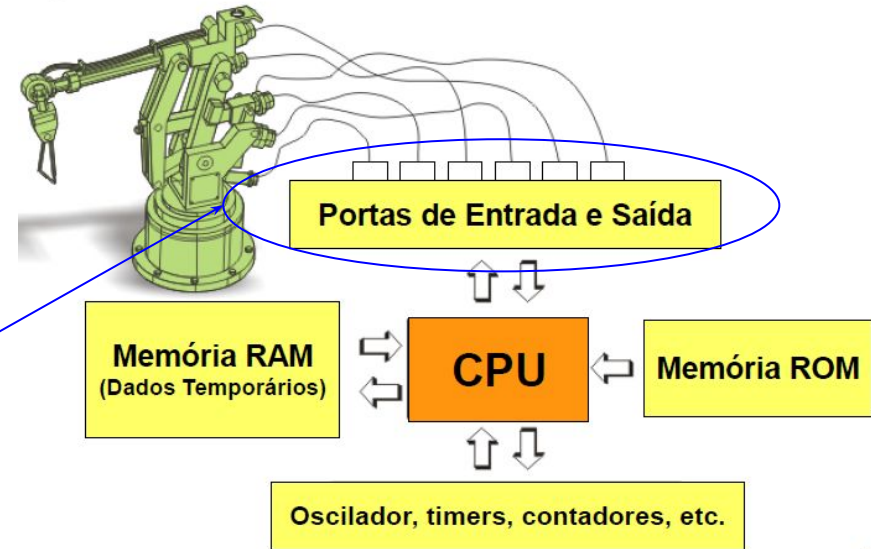
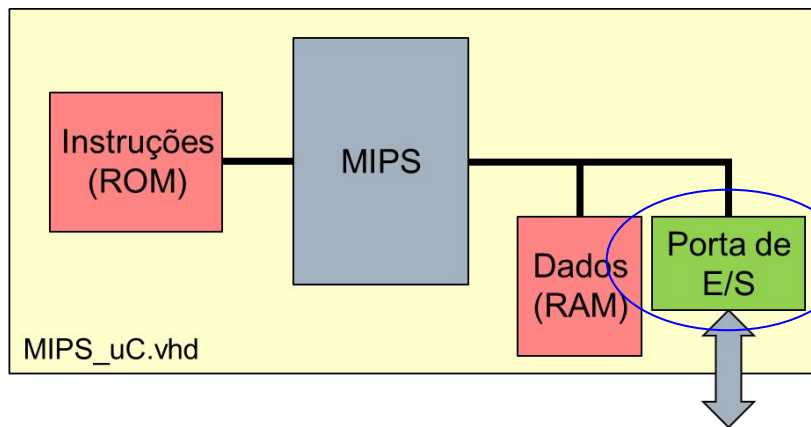
Nova entidade: MIPS\_uC.vhd



O objetivo do trabalho é adicionar ao sistema uma porta de E/S

# Interface entre Processador e Periféricos

- Estrutura básica de um sistema computacional
  - Porta de E/S adicionada



O objetivo do trabalho é adicionar ao sistema uma porta de E/S

# Interface entre Processador e Periféricos

---

## □ Trabalho 2

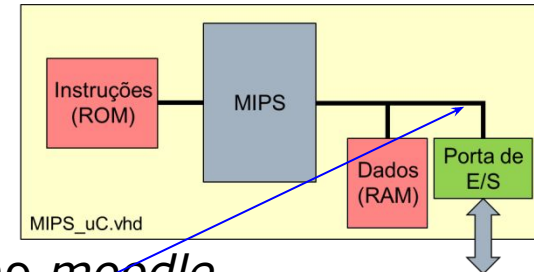
- A porta de E/S deve ter 16 bits para interface com o mundo externo
- Os bits da porta devem ser individualmente configuráveis como entrada ou saída
  - Exemplo
    - bits(15:10) e bits(1:0) - entrada
    - bits(9:2) - saída
- A configuração dos bits é controlada por um registrador (16 bits) onde cada bit corresponde à configuração de um bit da porta
  - 0: saída (*output*)
  - 1: entrada (*input*)
- Deve haver também um registrador que habilita individualmente a utilização dos bits da porta de E/S e um registrador de dados (ambos 16 bits)

# Interface entre Processador e Periféricos

## Trabalho 2

### Interface VHDL da porta de E/S

- Arquivo BidirectionalPort.vhd disponível no *moodle*
- Descrição VHDL pode ser comportamental, estrutural ou mista



```
entity BidirectionalPort is
```

```
generic (...);
```

```
port (
```

```
    clk      : in std_logic;
```

```
    rst      : in std_logic;
```

```
    -- Interface com o processador
```

```
    data_i   : in std_logic_vector (DATA_WIDTH-1 downto 0);
```

```
    data_o   : out std_logic_vector (DATA_WIDTH-1 downto 0);
```

```
    address  : in std_logic_vector (1 downto 0); -- NÃO ALTERAR!
```

```
    rw       : in std_logic; -- 0: read; 1: write
```

```
    ce       : in std_logic;
```

```
    -- Interface com o mundo externo
```

```
    port_io  : inout std_logic_vector (DATA_WIDTH-1 downto 0)
```

```
);
```

```
end BidirectionalPort;
```

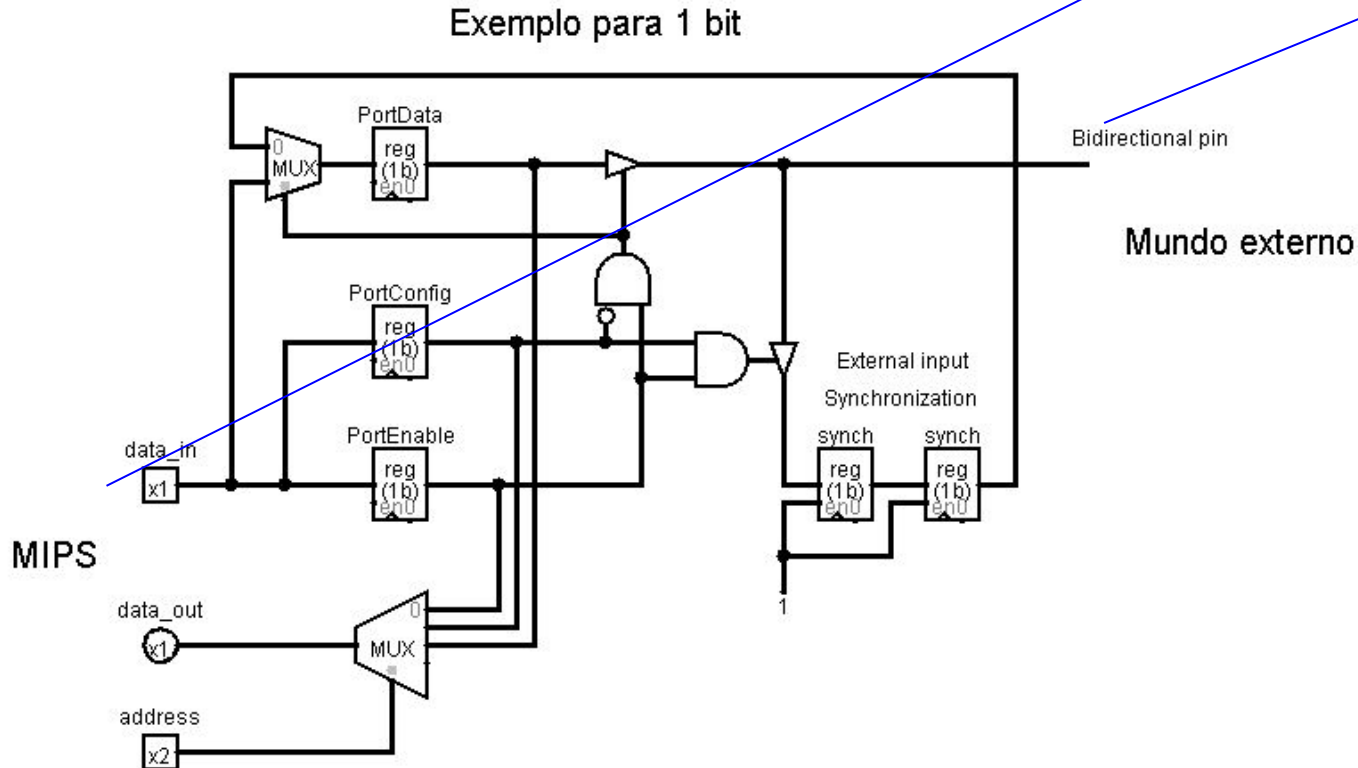
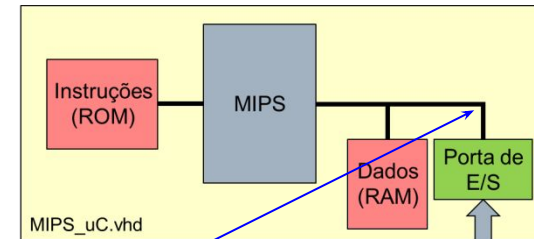
Esta interface não deve ser alterada!



# Interface entre Processador e Periféricos

## Trabalho 2

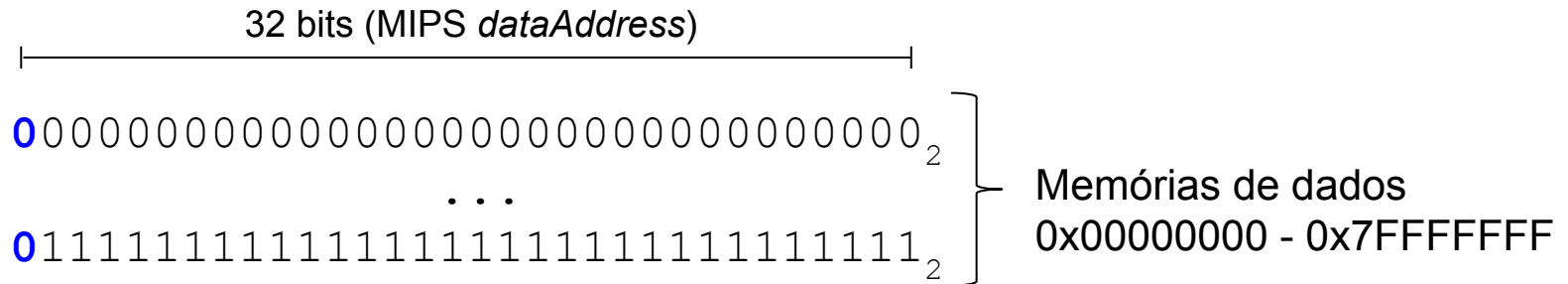
- Circuito resumido da porta de E/S
  - Descrição VHDL pode ser comportamental, estrutural ou mista



# Interface entre Processador e Periféricos

## Trabalho 2

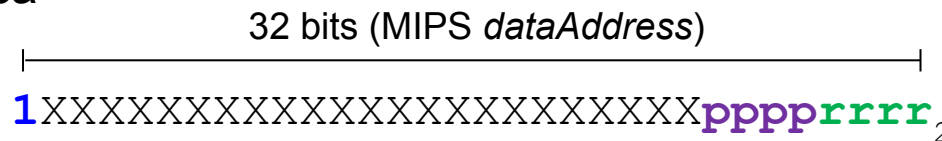
- Mapear os registradores da porta de E/S em memória
- Os endereços devem seguir o seguinte formato



MSb do endereço indica

0: memória

1: Entrada/Saída



ID do periférico  
(máximo 16)

Registradores dentro do periférico (máximo 16)

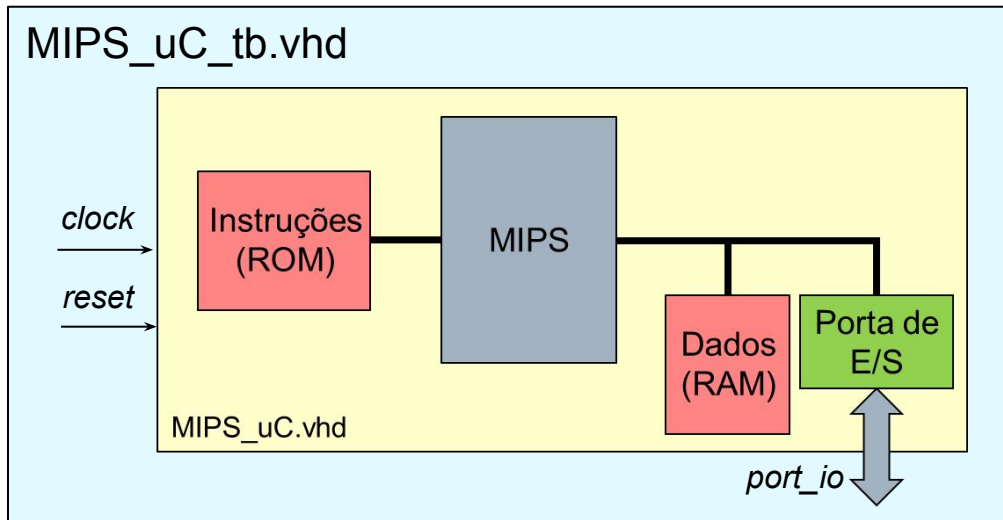
Escolher números para os registradores da porta de E/S e setar *generic* da *BidirectionalPort*

Escolher ID para Porta de E/S

# Interface entre Processador e Periféricos

## □ Trabalho 2

- A memória de dados deve ser desabilitada durante o acesso aos registradores da porta de E/S
- Estrutura dos arquivos VHDL
  - A interface do MIPS\_uC contém apenas *clock*, *reset* e *port\_io*
  - O *test bench* deve gerar o *clock*, *reset* e estímulos para os bits de *port\_io* que forem configurados como entrada

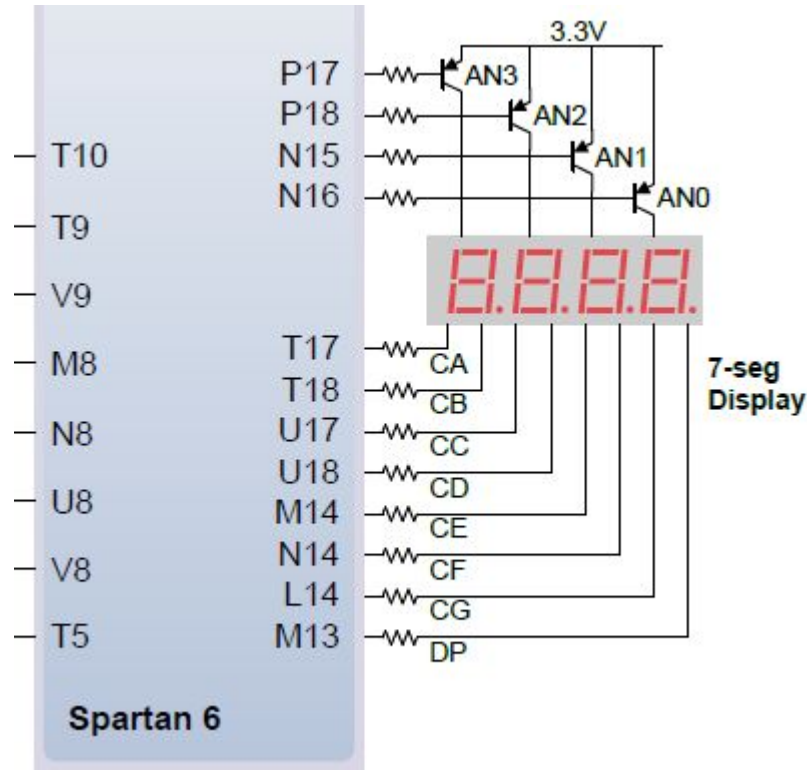


**Apresentar diagrama correspondente ao MIPS\_uC.vhd. Devem aparecer as ligações de todas interfaces do processador, memória e porta de E/S (*dataAddress*, *data\_i*, *data\_o*, *wbe*, *ce*, *rw* ...)**  
**Sugestão de editor : Dia Diagram Editor**

# Trabalho 2

---

- O MIPS\_uC deve controlar via **software** os *displays* da placa Nexys 3 através da(s) porta(s) de E/S
  - A porta de E/S deve ser ligada aos pinos do FPGA conectados aos *displays*



# Trabalho 2

---

## □ Aplicação

- Implementar um contador **hexadecimal** nos dois *displays* da direita
  - Manter os dois *displays* da esquerda apagados
- A contagem deve ser incrementada ou decrementada através de dois *push-buttons*
  - Cada vez que um *push-button* é pressionado o contador deve ser incrementado ou decrementado em **UMA** unidade
  - Utilizar **polling** nos *push-buttons* a fim de verificar seu estado



Atenção ao **flickering** dos *displays*!  
Verificar temporização dos *displays* no manual da placa

---

# Trabalho 2

---

## □ Aplicação

- Estruturar o código *assembly* em *main* e sub-rotinas
    - Utilizar JAL/JALR para chamar as subrotinas
    - Armazenar na pilha os registradores alterados por sub-rotinas e recuperá-los ao final (salvar/recuperar contexto)
    - Utilizar registradores de acordo com as convenções da arquitetura MIPS
  - Utilizar as instruções de *branch* e *jump* para implementar estruturas de controle de fluxo (*if/else, case*) e laços de repetição (*for, while*)
  - Dica: utilizar *array* para armazenar o código de 7 segmentos correspondentes aos números que devem ser mostrados nos *displays*
-