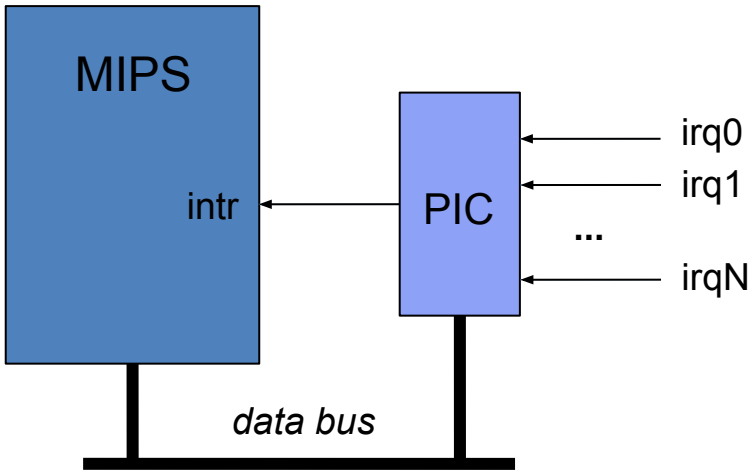
Trabalho 4 - parte 2

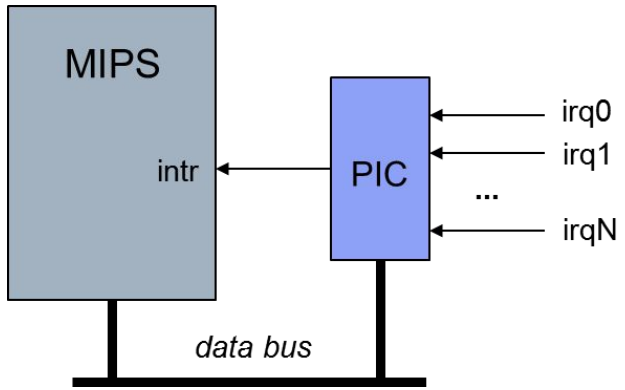
- Controlador de interrupções
  - Circuito que gerencia pedidos de interrupção de acordo com prioridades
  - Dá suporte ao mascaramento de pedidos interrupções



- Quando uma ou mais entradas de interrupção está ativa, o controlador gera uma interrupção para o processador
- O controlador resolve a prioridade das requisisões de interrupção (*irq*), quando várias estão ativas simultaneamente
- Ao interromper o processador, o controlador indica o número da interrupção a ser atendida (barramento de dados)
- Possui tipicamente 8 entradas de *irqs*
- Controladores de interrupções podem ser cascateados a fim de suportar mais *irqs*

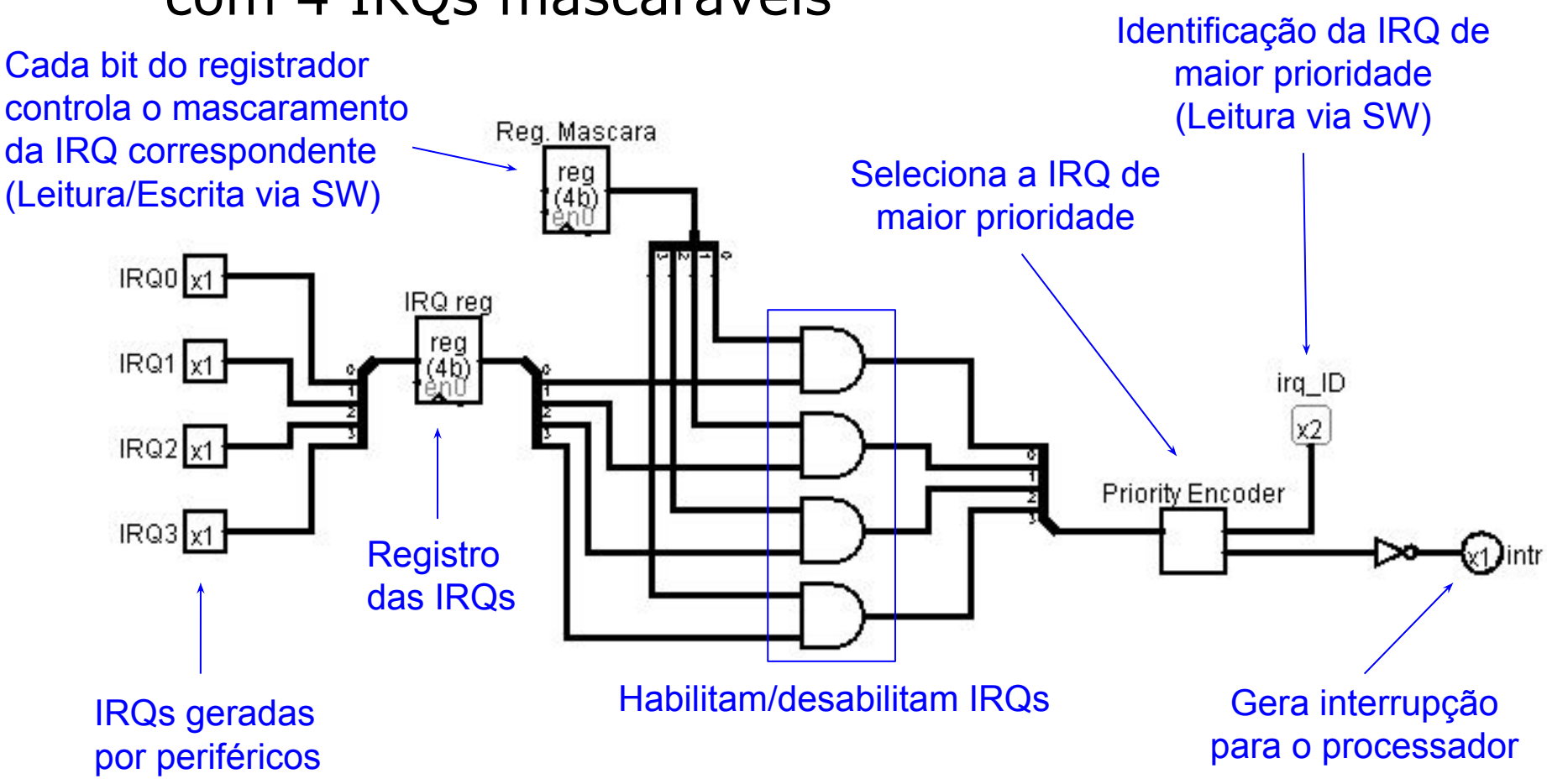
Trabalho 4 - parte 2

- Controlador de interrupções
  - Em certos momentos, durante a operação normal do processador pode ser necessário desabilitar todas interrupções ou apenas algumas (e.g. seção crítica)
  - Usa-se o termo mascarar quando se deseja desabilitar alguma(s) interrupção(ões) em específico
  - O mascaramento das interrupções é controlado por um registrador que contém a máscara de interrupções
    - Cada *bit* do registrador de máscara habilita/desabilita uma *irq*



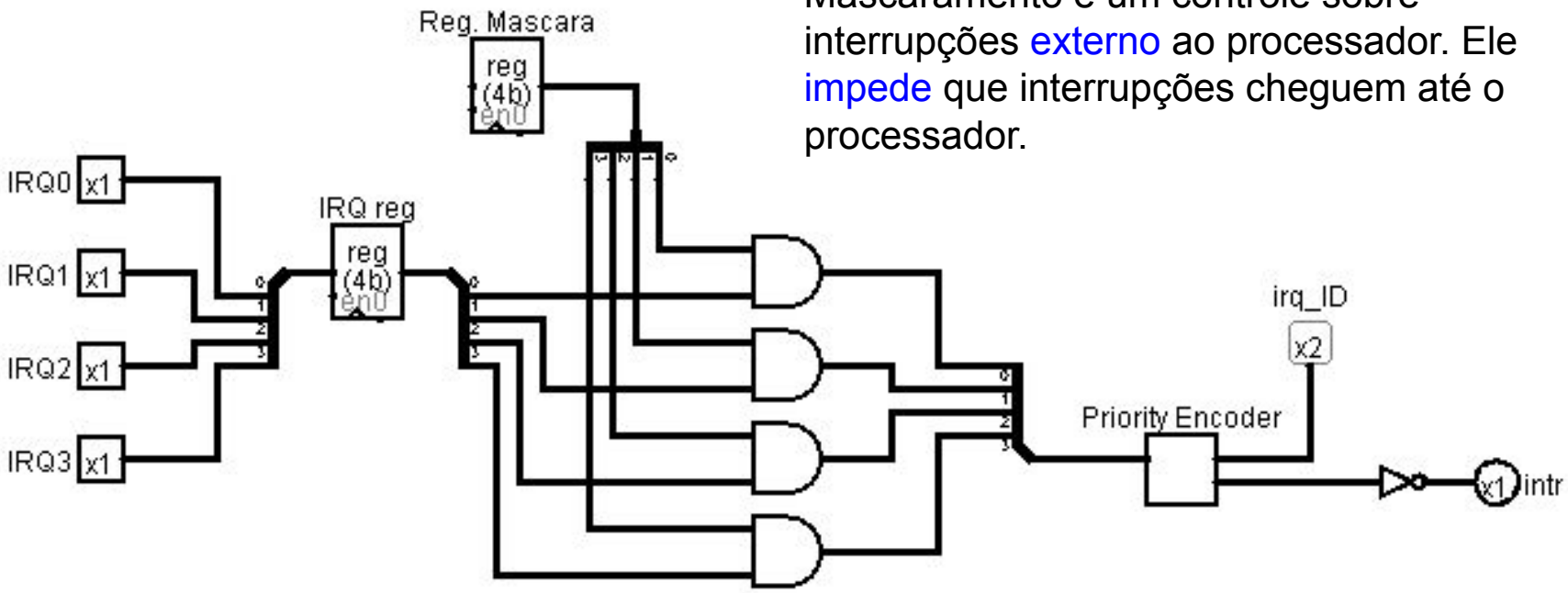
Trabalho 4 - parte 2

- Exemplo parcial de controlador de interrupções com 4 IRQs mascaráveis



Trabalho 4 - parte 2

- Exemplo parcial de controlador de interrupções com 4 IRQs mascaráveis

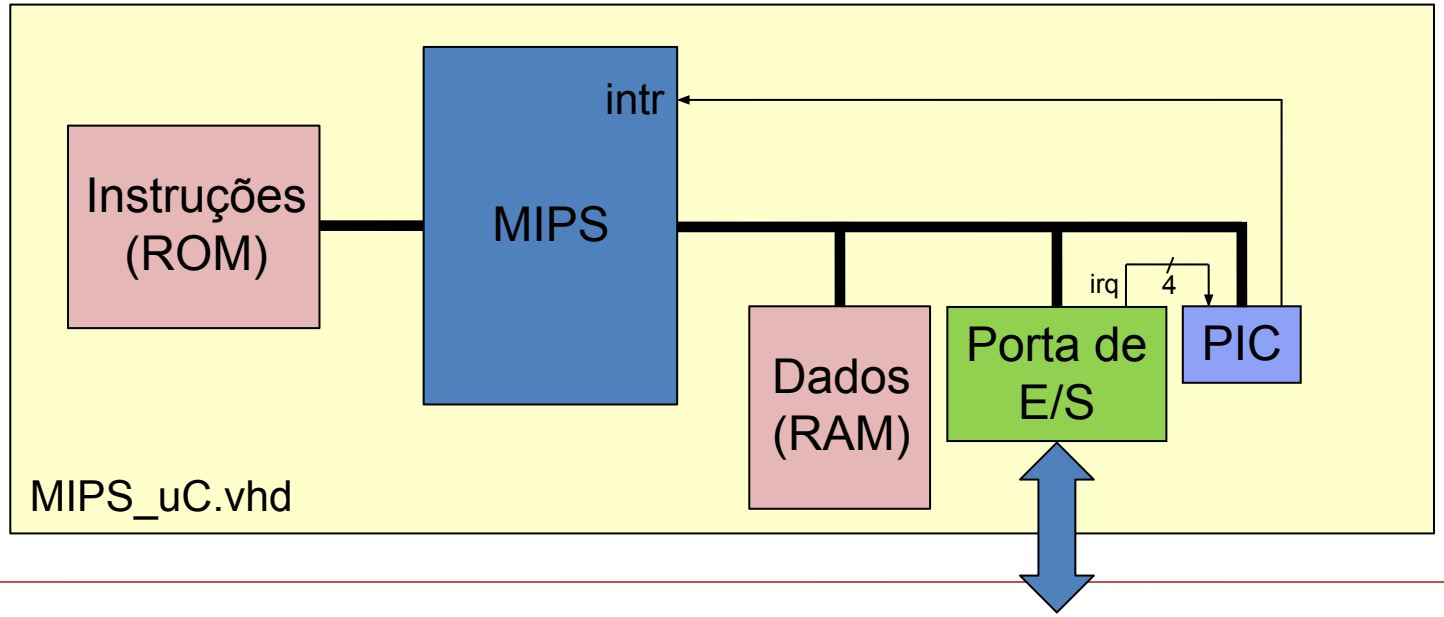


Mascaramento é um controle sobre interrupções externo ao processador. Ele impede que interrupções cheguem até o processador.

A desabilitação das interrupções é um controle interno ao processador. Ele ignora interrupções que chegam até o processador (mais seguro).

Trabalho 4 - parte 2

- Adicionar um controlador de interrupções (PIC) ao MIPS_µC
  - Os bits (15:12) da porta de E/S podem ser utilizados como entradas externas de interrupção e devem ser ligadas ao PIC
  - Utilizar as entradas de interrupção do PIC de 7:4 (menor prioridade)
  - As demais entradas de interrupção do PIC (3:0) devem ser mantidas em 0 (fazer isso no *port map* em MIPS_uC.vhd)



Trabalho 4 - parte 2

- Adicionar um controlador de interrupções (PIC) ao MIPS_µC
  - Moodle: InterruptController.vhd

```
entity InterruptController  is
    generic (
       IRQ_ID_ADDR: std_logic_vector(1  downto 0);-- Irq number address (Read)
       MASK_ADDR: std_logic_vector(1  downto 0);  -- Mask register address (R/W)
       INT_ACK_ADDR: std_logic_vector(1  downto 0)-- Interrupt ack address
(Write)
    );
    port (
        clk          : in std_logic;
        rst      : in std_logic;
        data         : inout std_logic_vector (7  downto 0);
        address      : in std_logic_vector (1  downto 0);
        rw           : in std_logic;   -- rw = 0: Read; rw = 1: Write
        ce           : in std_logic;
        intr         : out std_logic; -- To processor
        irq          : in std_logic_vector (7  downto 0) -- Interrupt request
    );
end InterruptController ;
```

Atenção
barramento bidirecional

Maior prioridade

Trabalho 4 - parte 2

- Tramento de interrupção vetorizada
  - Criar *array* e em cada posição armazenar o endereço de um *handler*

    int irq_handlers[8];
  - No *boot* ou na declaração, inicializar o *array* com os endereços dos *handlers*
    - No *boot*
      - irq_handlers[0] = endereço do *irq0_handler*
      - irq_handlers[1] = endereço do *irq1_handler*
      - *…*
    - Na declaração
      - irq_handlers .word irq0_handler irq1_handler …

Trabalho 4 - parte 2

- Tramento de interrupção vetorizada
  - Ao ser interrompido, o processador desvia para a ISR
  - Após o salvamento de contexto, deve-se ler do PIC o número da interrupção a ser tratada (IRQ_ID_ADDR)
  - Este número deve ser usado para indexar o *array* de endereços de *handlers* e recuperar o endereço do *handler* correspondente
    - Carregar endereço em um registrador e utilizar a instrução *jalr*
  - O processador então salta para o endereço do *handler* e executa a rotina
  - Após executar o *handler* e antes de recuperar o contexto, o processador deve notificar o PIC que a interrupção já foi tratada
    - Escrever em INT_ACK_ADDR o número da interrupção tratada
  - O PIC elimina a necessida de verificar todos os periféricos a fim de indentificar a origem da IRQ (*polled interrupt*)
    - O tempo para descobrir a origem passa a ser constante

Vetor da interrupção

Trabalho 4 - parte 2

- Tramento de interrupção vetorizada
  - Estrutura da ISR

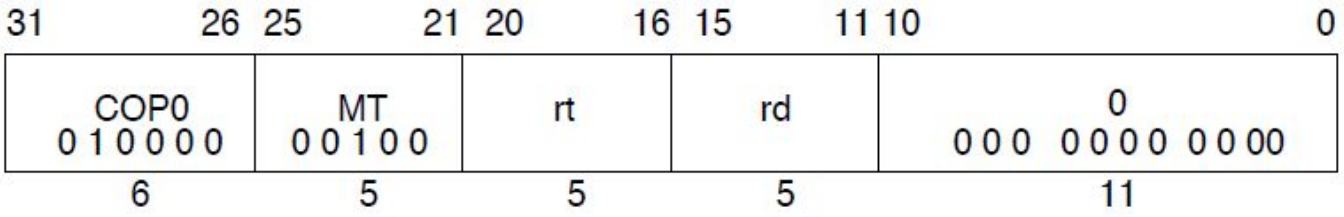
```
InterruptionServiceRoutine:
    1. Salvamento de contexto no PCB
   (todos regs menos 0, 26 e 27)
    2. Ler do PIC o número da IRQ
    3. Indexar irq_handlers e gravar em algum
       registrador o endereço do handler
    4. jalr reg (chama handler)
    5. Notificar PIC sobre a IRQ tratada
    6. Recuperação de contexto do PCB
    7. Retorno (ERET)

Periferico1Handler:
    ...
    jr $ra
...
```

Trabalho 4 - parte 2

- Instrução MTC0
  - Copia dados do banco de registradores do MIPS para o coprocessador 0 (CP0)
  - O CP0 é o circuito que dá suporte a funções do sistema operacional como manipulação de interrupções, gerenciamento de memória, escalonamento, etc

  - MTC0 *rt*, *rd*
    - *C0[rd]* ← *RF[rt]*

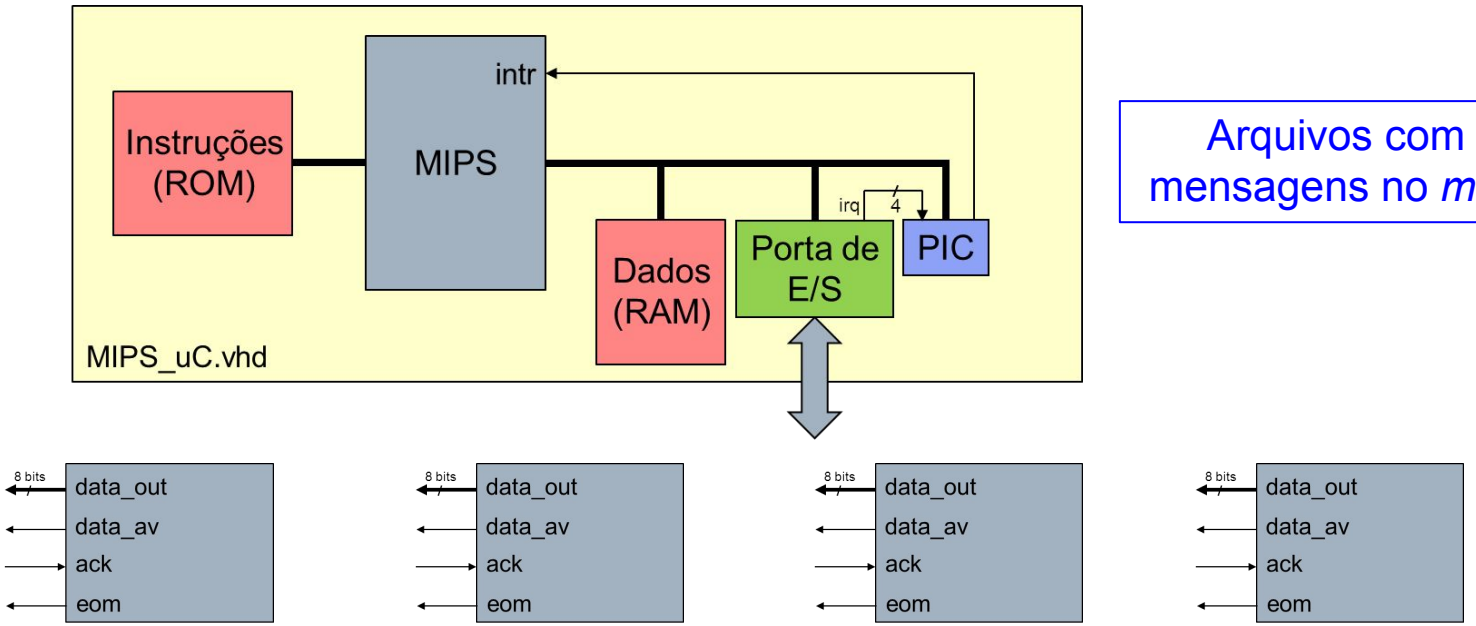


Trabalho 4 - parte 2

- Instrução MTC0
  - Adicionar ao *data path* um registrador que armazene o endereço da ISR
    - ISR_AD
  - O registrador deve ser escrito através da instrução MTC0
    - O registrador ISR_AD deve ser identificado como registrador $31 do CP0
  - Setar ISR_AD no *boot* com o endereço da ISR
    - *la $t0, InterruptServiceRoutine # Pega endereço do label*
    - *mtc0 $t0, $31 # ISR_AD ← $t0*

Trabalho 4 - parte 2

- Aplicação
  - Adicionar mais três *CryptoMessages*
    - Pode-se adicionar hardware externo ao MIPS_µC para a ligação
    - Os *CriptoMessages* devem gerar *irqs* (porta de E/S[15:12]) durante a execução do *Bubble Sort*
    - Apresentar diagrama da ligação



Arquivos com as mensagens no *moodle*

Trabalho 4 - parte 2

- Estrutura do código na memória