Sistemas Reconfiguráveis
Eng. de Computação

Profs. Francisco Garcia e Antônio Hamilton Magalhães

Aula 14 – Arquiteturas de processadores

Resposta ao exercício da aula 13

rev. A - 26/05/22

Exercício

Um micro controlador tem as seguintes características: arquitetura Harvard, um ciclo por instrução (exceto as instruções que carregam um novo valor no PC), 256 palavras de ROM com 11 bits de largura, 256 bytes de RAM e um registrador de 8 bits, chamado de **reg_a**. O seu conjunto de instruções possui as seguintes instruções:

Código	Mnem.	Descrição
`000 iiiiiiii`	ADD	Soma o valor presente no registrador reg_a com o valor imediato iiiiiiii. O resultado é armazenado no registrador reg_a.
`001 iiiiiiii`	AND	Faz a operação AND entre o valor presente no registrador reg_a com o valor imediato iiiiiiii. O resultado é armazenado no registrador reg_a.
`010 iiiiiiii`	OR	Faz a operação OR entre o valor presente no registrador reg_a com o valor imediato iiiiiiii. O resultado é armazenado no registrador reg_a.
`011 iiiiiiii`	XOR	Faz a operação XOR entre o valor presente no registrador reg_a com o valor imediato iiiiiiii. O resultado é armazenado no registrador reg_a.
`100 iiiiiiii`	MVI	Carrega no registrador reg_a o valor iiiiiiii.
`101 aaaaaaaa`	LDM	Carrega no registrador reg_a o conteúdo do endereço aaaaaaaa da RAM.
`110 aaaaaaaa`	STO	Armazena no endereço aaaaaaaa da RAM o conteúdo do registrador reg_a.
`111 aaaaaaaa`	JMP	Desvia a execução do programa para o endereço aaaaaaaa da memória de programa (ROM)

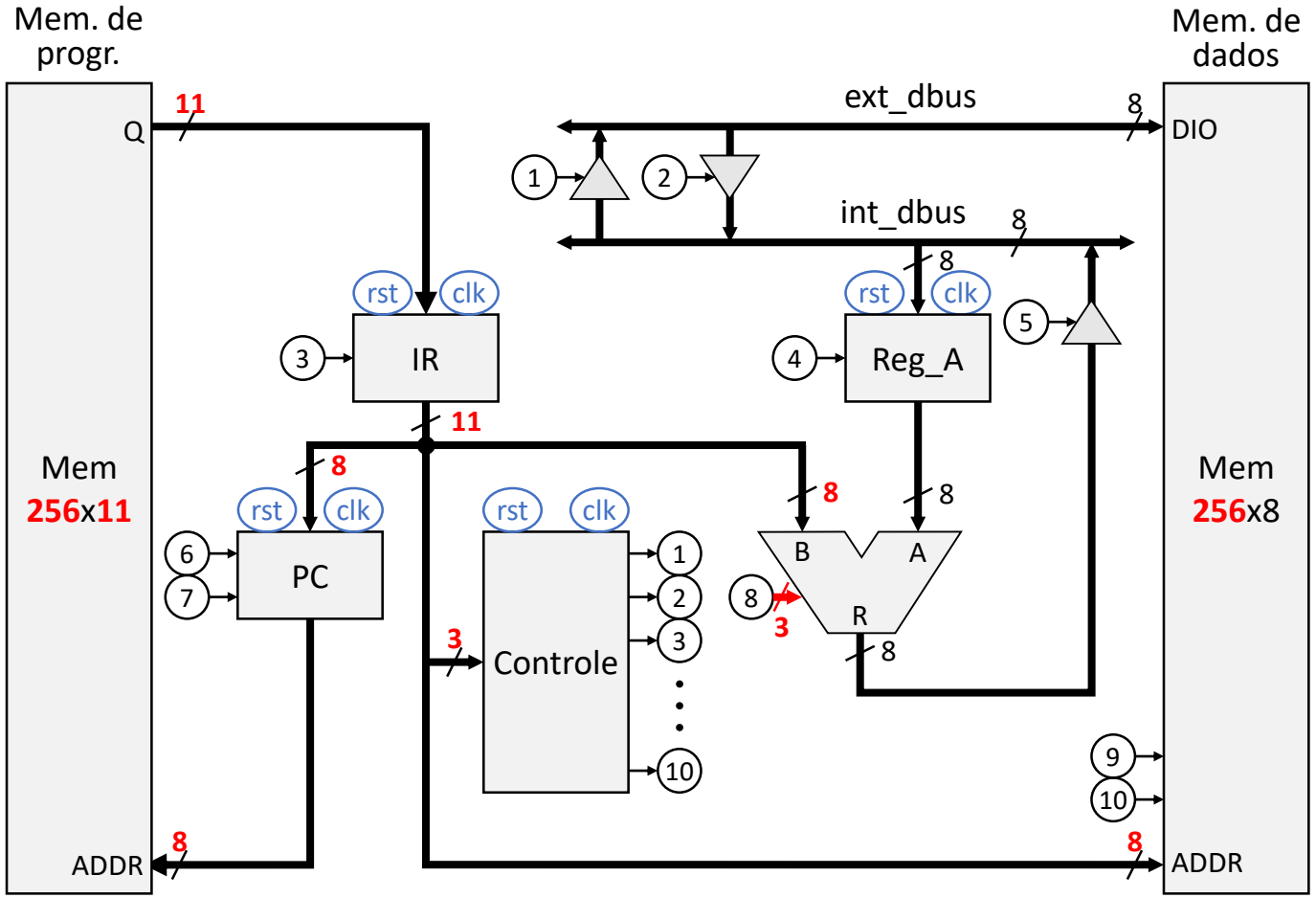
Exercício

Para esse processador:

1. Modifique o caminho de dados do exemplo da CPU somadora apresentado anteriormente, adequando a largura de cada barramento usado nesse caminho de dados ao exercício proposto;
2. Faça uma tabela das funções necessárias da ALU atribuindo valores binários a cada uma das funções.
3. Faça uma tabela com os valores de cada sinal de controle, para todos os estados da FSM e instruções.
4. Verifique quais os blocos do exemplo da CPU somadora podem ser aproveitados sem modificações e quais devem ser modificados.
5. Usando linguagem VHDL, crie os blocos necessários.
6. Faça a integração de todos os blocos e teste o projeto usando simulação funcional.

Exercício: Arquitetura Harvard

Exercício aula15 - Arquitetura interna (Harvard)

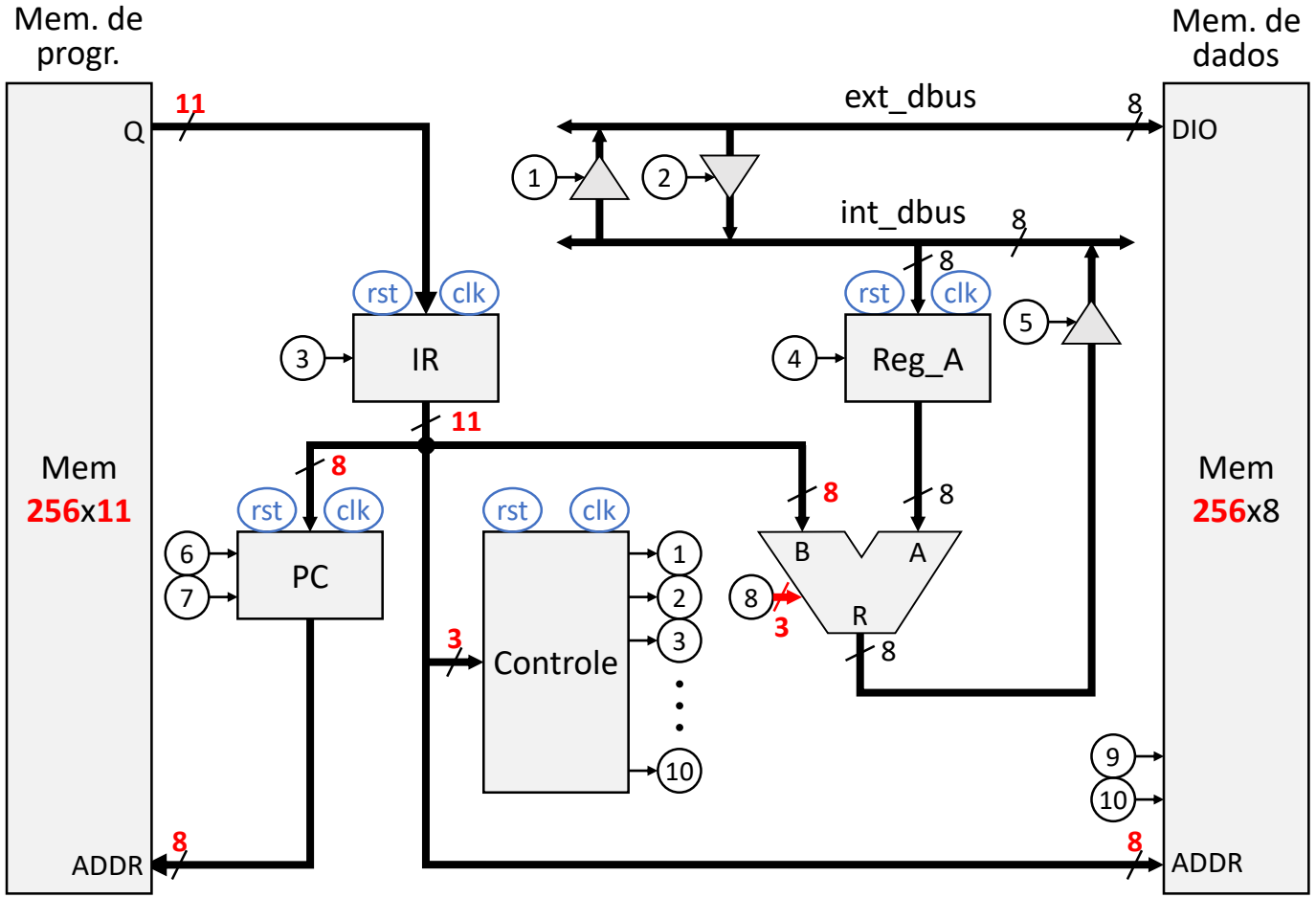


Sinal	Nome
1	dint_on_dext
2	dext_on_dint
3	wr_ir
4	wr_rega
5	au_on_dint
6	inc_pc
7	load_pc
8	**op_sel**
9	rd_mem
10	wr_mem

Todos os sinais ativos em nível alto, exceto **op_sel**

Exemplo: CPU somadora Arquitetura Harvard

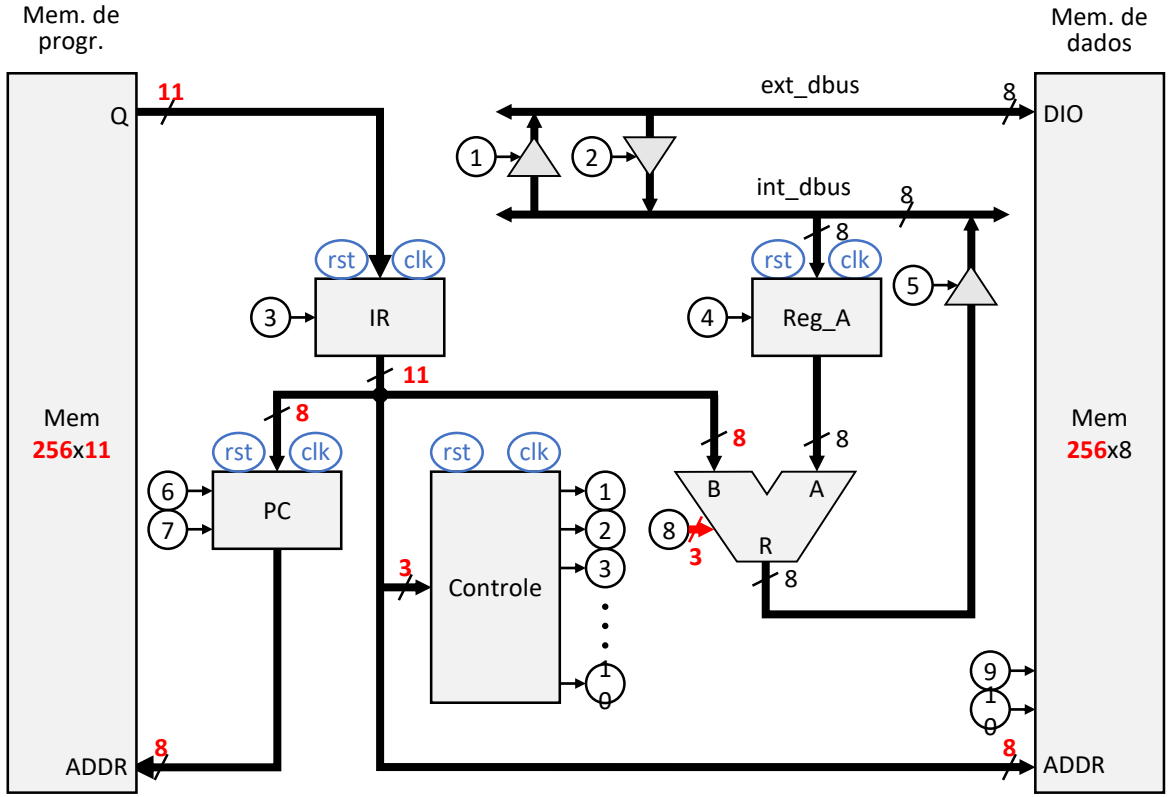
Exercício aula15 - Arquitetura interna (Harvard)



op_sel	operação
000	ADD
001	AND
010	OR
011	XOR
100	Pass_B
101	*don't care*
110	Pass_A
111	*don't care*

Exemplo: CPU somadora Arquitetura Harvard

Exercício aula15 - Arquitetura interna (Harvard)



Sinal	Nome	reset	fetch only	fetch_decod_exec							
				ADD	AND	OR	XOR	MVI	LDM	STO	JMP
1	dint_on_dext	0	0	0	0	0	0	0	0	**1**	0
2	dext_on_dint	0	0	0	0	0	0	0	**1**	0	0
3	wr_ir	0	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	0
4	wr_rega	0	0	**1**	**1**	**1**	**1**	**1**	**1**	0	0
5	au_on_dint	0	0	**1**	**1**	**1**	**1**	**1**	0	**1**	0
6	inc_pc	0	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	0
7	load_pc	0	0	0	0	0	0	0	0	0	**1**
8	op_sel	X	X	000	001	010	011	100	X	110	X
9	rd_mem	0	0	0	0	0	0	0	**1**	0	0
10	wr_mem	0	0	0	0	0	0	0	0	**1**	0

Exercício aula15 - controle

```
library ieee;
use ieee.std_logic_1164.all;
-------------------------------------------------------------------------
entity control_aula15 is
    port(
        nrst                : in std_logic;
        clk             : in std_logic;
        opcode          : in std_logic_vector(2 downto 0);
        dint_on_dext    : out std_logic;
        dext_on_dint    : out std_logic;
        au_on_dint: out std_logic;
        wr_rega         : out std_logic;
        wr_ir           : out std_logic;
        op_sel          : out std_logic_vector(2 downto 0);
        inc_pc          : out std_logic;
        load_pc         : out std_logic;
        rd_mem          : out std_logic;
        wr_mem          : out std_logic
    );
end entity;
```

Exercício aula15 - controle

```
architecture arch of control_aula15 is
    type state_type is     (rst, fetch_only, fet_dec_ex);
    signal pres_state      : state_type;
    signal next_state      : state_type;
    ------------------------------------------------------------
    constant instr_add     : std_logic_vector(2 downto 0) := "000";
    constant instr_and     : std_logic_vector(2 downto 0) := "001";
    constant instr_or      : std_logic_vector(2 downto 0) := "010";
    constant instr_xor     : std_logic_vector(2 downto 0) := "011";
    constant instr_mvi     : std_logic_vector(2 downto 0) := "100";
    constant instr_ldm     : std_logic_vector(2 downto 0) := "101";
    constant instr_sto     : std_logic_vector(2 downto 0) := "110";
    constant instr_jmp     : std_logic_vector(2 downto 0) := "111";
begin
    ------------------------------------------------------------
    -- parte sequencial da FSM
    ------------------------------------------------------------
    process(nrst, clk)
    begin
        if nrst = '0' then
            pres_state <= rst;
        elsif rising_edge(clk) then
            pres_state <= next_state;
        end if;
        end process;
```

Exercício aula15 - controle

```
    ------------------------------------------------------------
    -- parte sequencial da FSM
    ------------------------------------------------------------
    process(nrst, clk)
    begin
        if nrst = '0' then
            pres_state <= rst;
        elsif rising_edge(clk) then
            pres_state <= next_state;
        end if;
        end process;
```

Exercício aula15 - controle

```
-------------------------------------------------------------
-- parte combinacional da FSM
-------------------------------------------------------------
process(nrst, pres_state, opcode)
begin
    -- valores default:
    dint_on_dext  <= '0';
    dext_on_dint  <= '0';
    au_on_dint    <= '0';
    wr_rega       <= '0';
    wr_ir         <= '0';
    op_sel        <= "---";
    inc_pc        <= '0';
    load_pc       <= '0';
    rd_mem        <= '0';
    wr_mem        <= '0';

    case pres_state is
        when rst =>
            next_state <= fetch_only;

        when fetch_only =>
            next_state <= fet_dec_ex;
            wr_ir        <= '1';
            inc_pc       <= '1';
```

Exercício aula15 - controle

```
when fet_dec_ex =>
    case opcode is
    ------------------------------------------
    -- add, and, or, xor, mvi
    ------------------------------------------
        when instr_add | instr_and | instr_or |
             instr_xor | instr_mvi =>

            next_state <= fet_dec_ex;
            wr_ir      <= '1';
            wr_rega    <= '1';
            au_on_dint <= '1';
            inc_pc     <= '1';
            case opcode is
                when instr_add => op_sel <= "000";
                when instr_and => op_sel <= "001";
                when instr_or  => op_sel <= "010";
                when instr_xor => op_sel <= "011";
                when instr_mvi => op_sel <= "100";
                when others => null;
            end case;
```

Exercício aula15 - controle

```
-------------------------------------------
-- ldm
-------------------------------------------
    when instr_ldm =>
        next_state    <= fet_dec_ex;
        dext_on_dint  <= '1';
        wr_ir         <= '1';
        wr_rega       <= '1';
        inc_pc        <= '1';
        rd_mem        <= '1';
-------------------------------------------
-- sto
-------------------------------------------
    when instr_sto =>
        next_state    <= fet_dec_ex;
        dint_on_dext  <= '1';
        wr_ir         <= '1';
        au_on_dint    <= '1';
        inc_pc        <= '1';
        wr_mem        <= '1';
```

Exercício aula15 - controle

```
                -------------------------------------------
                -- jmp
                -------------------------------------------
                    when instr_jmp =>
                        next_state     <= fetch_only;
                        load_pc        <= '1';
                end case;
            end case;
        end process;

end arch;
```

Fim