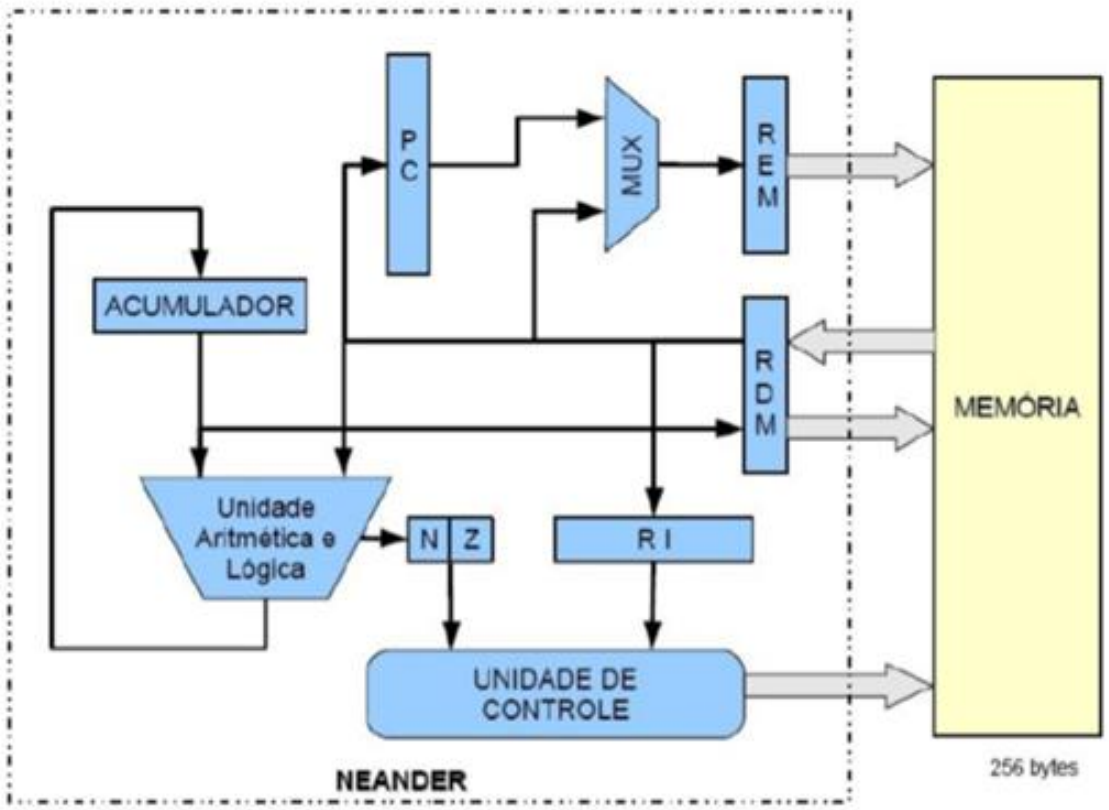
Neander – Mult + BEQ + BNQ

Autores: Patrícia Teixeira Davet e Thiago Ferreira Pontes
Professores: Rafael Soares e Bruno Zatt
Instituição: UFPEL – PPGC

Julho de 2014

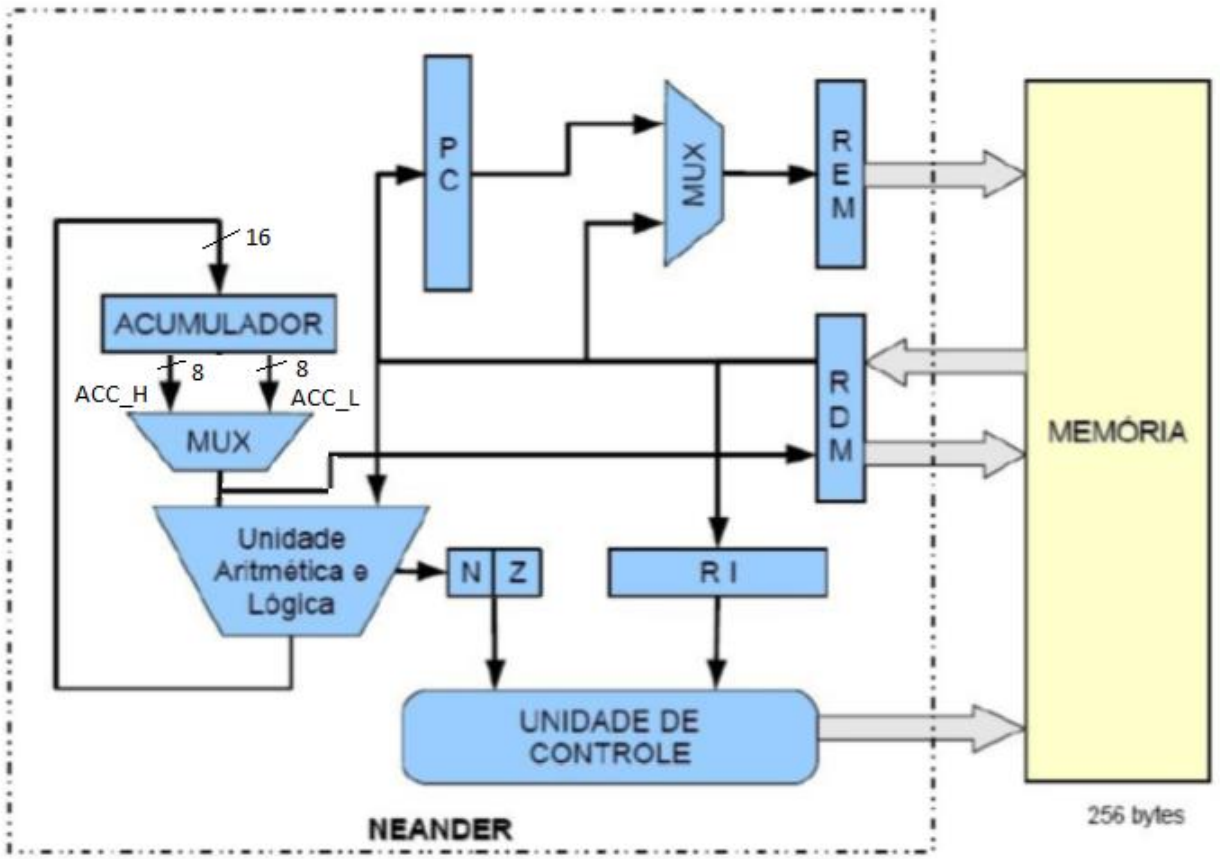
Neander

Arquitetura Neander: Nível RT



Neander – Instrução Multiplicação

Arquitetura Neander: Nível RT



Neander – Instrução Multiplicação Unidade Lógica e Aritmética

- Alteração no número de bits de saída da ULA para 16

```
entity ULA is
    port(
            a,b         : in std_logic_vector(7 downto 0);  -- bits de entrada da ULA
            sel         : in std_logic_vector(2 downto 0);  -- bits de selecao de funcao
            zero        : out std_logic;                    -- bit verificacao se zero
            negativo    : out std_logic;                    -- bit verificacao se negativo
            saida       : out std_logic_vector (15 downto 0)-- bits de saidas da ULA
        );
end ULA;
```

- Criação do sinal saida_mult

```
-- ------------inicio da declaracao dos sinais ----------------------------------------
signal saida_somador    : std_logic_vector (7 downto 0); -- bits de saida do somador
signal saida_subtrator  : std_logic_vector (7 downto 0); -- bits de saida do subtrator
signal saida_mult       : std_logic_vector (15 downto 0);-- bits de saida do multiplicador
signal saida_not        : std_logic_vector (7 downto 0); -- bits de saida do inversor A
signal saida_or         : std_logic_vector (7 downto 0); -- bits de saida logica or
signal saida_and        : std_logic_vector (7 downto 0); -- bits de saida logica and
signal saida_aux        : std_logic_vector (15 downto 0);-- bits de saida da ULA
-- ================================================================================
```

Neander – Instrução Multiplicação Unidade Lógica e Aritmética

- Adição da operação de multiplicação e concatenação com 8 bits em zero, quando a operação não for multiplicação

```
begin
    soma        :som_sub port map(      -- realiza a soma
                            a => a,
                            b => b,
                            s => saida_somador
                        );

    saida_not <= not a;                 -- realiza a logica de inversao
    saida_or  <= a or b;                -- realiza a logica or
    saida_and <= a and b;               -- realiza a logica and
    saida_subtrator <= a-b;             -- realiza a operacao de subtracao
    saida_mult <= a * b;                -- realiza a operacao de multiplicacao

    with sel select    -- Operação                      -- Função
        saida_aux <=  "00000000" & saida_somador     when "000", -- A+B
                      "00000000" & saida_and         when "001", -- A and B
                      "00000000" & saida_or          when "010", -- A or B
                      "00000000" & saida_not         when "011", -- not A
                      "00000000" & saida_subtrator   when "100", -- A-B
                                   saida_mult        when "101", -- A*B
                      "00000000" & b                 when others;-- B
```

Neander – Instrução Multiplicação Operação

- Mapeamento do Acumulador e MUX seletor dos 8 bits mais ou menos significativos do Acumulador

```
-- mapeamento do Acumulador ---------------------------------------
                ACC: ACUMULADOR port map(
                        entrada     => saida_ULA,
                        saida       => entrada_mux_ACC,
                        carga       => ACC_STORE,
                        reset       => broadcast_reset
                    );
            with ACC_MUX_ADDRESS select
            entrada_RDM <= entrada_mux_ACC(15 downto 8) when '1',
                           entrada_mux_ACC(7 downto 0)   when '0';
```

Neander – Instrução Multiplicação Operação

```
-- mapeamento do BCD
    bcd_dec_acc_1: bcd_decode port map(
                        bcd_in  => entrada_RDM(3 downto 0),
                        bcd_out => ACC_OUT_L
        );
    bcd_dec_acc_2: bcd_decode port map(
                        bcd_in  => entrada_RDM(7 downto 4),
                        bcd_out => ACC_OUT_H
        );
    bcd_dec_acc_X: bcd_decode port map(
                        bcd_in  => entrada_mux_ACC(11 downto 8),
                        bcd_out => ACC_OUT_L_2
        );
    bcd_dec_acc_Y: bcd_decode port map(
                        bcd_in  => entrada_mux_ACC(15 downto 12),
                        bcd_out => ACC_OUT_H_2
        );
    bcd_dec_pc_1: bcd_decode port map(
                        bcd_in  => entrada_mux_PC(3 downto 0),
                        bcd_out => PC_OUT_L
        );
    bcd_dec_pc_2: bcd_decode port map(
                        bcd_in  => entrada_mux_PC(7 downto 4),
                        bcd_out => PC_OUT_H
        );

broadcast_reset <= reset;
```

Neander – Instrução Multiplicação Unidade de Controle

```
-- IDENTIFICA A INSTRUCAO ------------------------------------------------------
                when ST_RI_READ =>
                    REM_ST      <= '1';                 -- carrega REM <- MEM(PC+1)
                    RI_ST       <= '0';
                    PC_INCRE    <= '0';
                       if RI_IN = "00000000" then       -- instrucao NOP 00
                        estado <= ST_NOP;
                    elsif RI_IN = "00010000" then       -- instrucao STA 10
                        estado <= ST_STA;
                    elsif RI_IN = "00100000" then       -- instrucao LDA 20
                        estado <= ST_LDA;
                    elsif RI_IN = "00110000" then       -- instrucao ADD 30
                        estado <= ST_ADD;
                    elsif RI_IN = "01000000" then       -- instrucao OR  40
                        estado <= ST_OR;
                    elsif RI_IN = "01010000" then       -- instrucao AND 50
                        estado <= ST_AND;
                    elsif RI_IN = "01100000" then       -- instrucao NOT 60
                        estado <= ST_NOT;
                    elsif RI_IN = "01110000" then       -- instrucao MULT 70
                        estado <= ST_MULT;
                    elsif RI_IN = "10000000" then       -- instrucao JMP 80
                        estado <= ST_JMP;
                    elsif RI_IN = "10010000" then       -- instrucao JN 90
                        estado <= ST_JN;
                    elsif RI_IN = "10100000" then       -- instrucao JZ A0
                        estado <= ST_JZ;
                    elsif RI_IN = "10110000" then       -- instrucao BEQ B0
                        estado <= ST_BEQ;
                    elsif RI_IN = "11000000" then       -- instrucao BNQ C0
                        estado <= ST_BNQ;
                    elsif RI_IN = "11110000" then       -- instrucao HALT F0
                        estado <= ST_HALT;
                    end if;
```

Neander – Instrução Multiplicação Unidade de Controle

```
-- -------------INSTRUCAO MULT-------------------------------------------------------
                when ST_MULT =>
                    flag_bnq <= '0';
                    flag_beq <= '0';
                    flag_mult  <= '1';
                    MUX_ADD     <= '1';                 -- MUX = RDM(PC-ant)
                    MUX_ADD_2   <= '0';
                    REM_ST      <= '0';
                    ULA_CTRL    <= "101";               -- ULA em MULT
                    estado      <= ST_RDM_ATUALIZA;

-- ================================================================================
```

```
                when ST_MEM_STORE =>
                   REM_ST       <= '0';
                    RDM_ST      <= '0';
                    PC_INCRE    <= '1';
                    if flag_mult = '0' then
                        estado      <= ST_MUX_ADD;
                    else -- guardar parte alta
                        flag_mult  <= '0';
                        estado      <= ST_STA_2;
                    end if;
```

Neander – Instruções BEQ e BNQ

- Instrução BEQ (*Branch on equal*) –
  - Formato: BEQ *End_0 End_1*
    - BEQ – Mnemônico da instrução;
    - *End_0* – Endereço de memória que contém o valor a ser comparado;
    - *End_1* – Endereço de desvio.
  - Descrição: O programa desviará para o endereço *End_1* caso o conteúdo do acumulador seja igual ao conteúdo da posição de memória indicada por *End_0*.
  - Sequência de operações:
    - Busca:
      - RI <- MEM(PC);
      - PC <- PC+1;
    - Execução:
      - end <- MEM(PC);
      - PC <- PC+1;
      - AC – MEM(end);
      - end <- MEM(PC);
      - Se (Z = 1) -> PC <- end;
      - Se (Z = 0) -> PC <- PC+1.

Neander – Instruções BEQ e BNQ

- Instrução BNQ (*Branch on not equal*) –
  - Formato: BNQ *End_0 End_1*
    - BNQ – Mnemônico da instrução;
    - *End_0* – Endereço de memória que contém o valor a ser comparado;
    - *End_1* – Endereço de desvio.
  - Descrição: O programa desviará para o endereço *End_1* caso o conteúdo do acumulador seja diferente do conteúdo da posição de memória indicada por *End_0*.
  - Sequência de operações:
    - Busca:
      - RI <- MEM(PC);
      - PC <- PC+1;
    - Execução:
      - end <- MEM(PC);
      - PC <- PC+1;
      - AC – MEM(end);
      - end <- MEM(PC);
      - Se (Z = 0) -> PC <- end;
      - Se (Z = 1) -> PC <- PC+1.

Neander – Programa de Teste

- Objetivo – Testar o funcionamento das instruções implementadas.
  - Programa:

```
RI   I
0 - LDA
1 - 129 (0X02)
2 - ADD
3 - 130 (0X01)   2+1 = 3 (0X03)
4 - BEQ
5 - 131 (0X0A)
6 - 111
7 - ADD
8 - 129 (0X02)   3+2 = 5 (0X05)
9 - BEQ
10- 131 (0X0A)
11- 111
12- MULT
13- 129 (0X02)   5*2 = 10 (0X0A)
14- BEQ
15- 131 (0X0A)
16- 111 (0X6F)
17- HALT
.
.
.
.
.
.
111- ADD
112- 130 (0X01)   while (AC < 15) AC = AC+1;
113- BNQ
114- 132 (0X0F)
115- 111
116- MULT
117- 133 (0XFF)      15*255 = 3825 (0X0EF1)
118- HALT
```

Neander – Mult + BEQ + BNQ

Autores: Patrícia Teixeira Davet e Thiago Ferreira Pontes
Professores: Rafael Soares e Bruno Zatt
Instituição: UFPEL – PPGC

Julho de 2014