Departamento de CIÊNCIA DA COMPUTAÇÃO

UFRR

INSALUBYTE

Processador 16 bits, RISC, MIPS

Alunos: Victor Hugo, Giovana Oliveira e Ryan Kayky
Disciplina: Arquitetura e Organização de Computadores
Professor: Herbert Oliveira Rocha
Semestre: ERE 2023.2
30 de Novembro de 2023
Boa Vista - RR

CARACTERISTICAS DO PROCESSADOR

1. Processador 16 bit, 65 536 linhas de código em um programa
2. 16 bits de espaço na memória RAM;
3. 16 registradores disponíveis
4. O J realiza um salto de 4095 linhas

TIPO DE INSTRUÇÕES

Tipo R

4 bits	4 bits	4 bits	4 bits
15-12	11-8	7-4	3-0
Opcode	Reg1	Reg2	Reg3

- Este formato aborda instruções instruções baseadas em operações aritméticas, como add, sub e mult.

COMPONENTES

1. ULA or ALU e Branch Helper
2. Somador, PC, Divisor de instruções, Extensor de sinal 4x16 bits
3. Unidade de controle UC
4. Memória RAM
5. Banco de registradores
6. Multiplexador

COMPONENTES

PC

- Trecho do código do somador

```
1   LIBRARY IEEE;
2   USE IEEE.STD_LOGIC_1164.ALL;
3
4   ENTITY PC IS
5       PORT (
6           clock: in std_logic; -- sinal do clock
7           enderecoDEentrada: in std_logic_vector (15 downto 0); --Sinal do enderço
8           enderencoDEsaida:  out std_logic_vector (15 downto 0)
9           ); --Sinal da saida do endereço
10  END PC;
11
12  architecture BEHAVIOR of PC is
13  BEGIN
14      process (clock)
15      BEGIN
16          IF RISING_EDGE(clock) THEN
17              enderencoDEsaida <= enderecoDEentrada;
18          END IF;
19      END PROCESS;
20  END;
21
```

COMPONENTES

SOMADOR

- Trecho do código do somador

```
1   --somador
2   --PC
3   --Bibliotecas e pacotes
4   LIBRARY IEEE;
5   USE IEEE.std_logic_1164.all;
6   USE IEEE.STD_LOGIC_UNSIGNED.ALL;
7
8   --Entidade
9   entity somador is
10      port(
11      Clock:      in std_logic;
12      entrada:    in std_logic_vector(15 downto 0);
13      saida:      out std_logic_vector(15 downto 0)
14  );
15  end entity;
16
17  architecture Behavior of somador is
18  begin
19      --Quando clock subir para nivel alto
20      --a saida vai pegar a proxma linha de endereço
21      process(clock, entrada)
22      begin
23          saida <= entrada + '1';
24      end process;
25  end architecture;
26
```

COMPONENTES

DIVISOR DE INSTRUÇÕES

- Trecho do código do divisor

```
1   --Divisor
2   --Divisor da instruções da rom
3   LIBRARY IEEE;
4   USE IEEE.STD_LOGIC_1164.ALL;
5
6   entity Divisor is
7       port(
8       instrucao:                  in std_logic_vector(15 downto 0);
9       opcode, rs, rt, rd:         out std_logic_vector(3 downto 0);
10      endereco:                   out std_logic_vector(15 downto 0)
11      );
12  end entity;
13  architecture behavior of Divisor is
14  begin
15      opcode <= instrucao(15 downto 12);
16      rs <= instrucao(11 downto 8);
17      rt <= instrucao(7 downto 4);
18      rd <= instrucao(3 downto 0);
19      endereco(11 downto 0) <= instrucao(11 downto 0);
20      endereco(15 downto 12) <= (others =>'0');
21  end architecture;
22
```

COMPONENTES

EXTENSOR DE SINAL 4x16bits

- Trecho do código do extensor de sinal 4xbits

```
--Extensor 4x16
--Bibliotecas e pacotes
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
--Entidade
entity Extensor4x16 is
    port(
    entrada:    in std_logic_vector(3 downto 0);
    saida:      out std_logic_vector(15 downto 0)
    );
end entity;
architecture Behavior of Extensor4x16 is
begin
    process(entrada)
    begin
        saida(3 downto 0) <= entrada;
        saida(15 downto 4) <= (others =>'0');
    end process;
end architecture;
```

COMPONENTES

MEM. ROM

- Trecho do código da ROM

```
21
22  -- Entidade
23  entity Rom is
24      port(
25          Clock    : in std_logic;
26          endereco : in std_logic_vector(15 downto 0);
27          saida    : out std_logic_vector(15 downto 0)
28      );
29  end entity;
30
31  architecture Behavior of Rom is
32  type memoria_array is array(natural range <>) of std_logic_vector(15 downto 0);
33  constant operacoes : memoria_array(0 to 17) :=
34  (
35  --    Testes Sub e Add| funcionando
36  --    0 => "0001000000000011", -- Addi S0 S0 3
37  --    1 => "0001000100010001", -- Addi S1 S1 1
38  --    2 => "0011000000000001", -- Subi S0 S0 1
39  --    3 => "0010001000000001", -- Sub S2 S0 S1
40
41  --    Testes Beq e Li
42  --    0 => "0110000000000010",    -- Li S0 2
43  --    1 => "0110000100010010",    -- Li S1 2
44  --    2 => "1000000000000001",    -- If S0 == S1
45  --    3 => "0111000000010101",    -- Beq S0 == S1 Jump 0101
46  --    4 => "0001000000000001",    -- Addi S0 S0 1
47  --    5 => "0001000000000010",    -- Addi S0 S0 2
48
49  --    Teste fibonacci
50        0 => "0001000000000000", -- Addi S0 0
51        1 => "0101000000000000", -- Sw S0
52        2 => "0001000000000001", -- Addi S0 1
53        3 => "0001000100010001", -- Addi S1 1
54        4 => "0100001000100000", -- Lw S2 00
55        5 => "0000001000100001", -- Add S2 S1
56        6 => "0000000100010000", -- Add S1 S0
57        7 => "0100000000000000", -- Lw S0 00
58        8 => "0000000000000010", -- Add S0 S2
59        9 => "1001000000000100", -- J 0100
60
61  ----  Teste fatorial fat
62  --    0 => "0110000000001000",    -- n Li S0 0
```

TIPO DE INSTRUÇÕES

Tabela geral de instruções

Opcode	Nome	Formato	Breve Descrição	Exemplo
0000	ADD	R	Soma	**add** $S0, $S1, $S2, ou seja, $S0 := $S1+$S2
0001	ADDi	I	Soma Imediata	**addi** $S0, $S1, X, ou seja, $S0 := $S1+X
0010	SUB	R	Subtração	**sub** $S0, $S1, $S2, ou seja, $S0 := $S1 - $S2
0011	SUBi	I	Subtração Imediata	**subi** $S0, $S1, X, ou seja, $S0 := $S1-X
0100	LW	I	Load Word	**lw** $S0, 0($S0)
0101	SW	I	Store Word	**sw** $S0, 0($S0)
0110	LI	I	Load Imediato	**li** $S0, 31
0111	BEQ	R	Salto Condicional	**beq** $S0, $S1, L1
1000	IF	R	Condição	**If** $S0, $S1
1001	J	J	Salto	**J** L1
1010	MULT	R	Multiplicação	**mult** $S0, $S1, $S2, ou seja, $S0 := $S1*$S2
1011	MULTi	I	Multiplicação imetiada	**multi** $S0, $S1, X, ou seja, $S0 := $S1*X

COMPONENTES

ULA

- Trecho do código da ula

```
--Entidade
entity Ula is
PORT(
        Clock:          in std_logic;
        AluOp:          in std_logic_vector(3 downto 0);
        rs,rd:          in std_logic_vector(15 downto 0);
        resultado:      out std_logic_vector(15 downto 0);
        z:              out std_logic
   );
end entity;
architecture Behavior of Ula is
Signal in_branch_helper, out_branch_helper: std_logic;

component Branch_helper is
   port(
      A: in std_logic;
      S: out std_logic
);
end component;
begin

   Bh: Branch_helper port map(in_branch_helper, out_branch_helper);

   process (Clock, AluOp, rs, rd, in_branch_helper, out_branch_helper)
   begin
      case AluOp is

         when "0000" | "0001" => --add
            resultado <= rs + rd;

         when "0010" | "0011" => --sub
            resultado <= rs - rd;

         when "0100" => --lw
            resultado <= rs;

         when "0101" => --sw
            resultado <= rs;

         when "0110" => --li
            resultado <= rd;
```

COMPONENTES

ULA

- Trecho do código da ula

```
50          WHEN "0111" => --Beq
51              if out_branch_helper = '1' then
52                  z <= '1';
53              else
54                  z <= '0';
55              end if;
56              resultado <= "0000000000000000";
57
58          when "1000" => -- if
59              if rs = rd then
60                  in_branch_helper <= '1';
61              else
62                  in_branch_helper <= '0';
63              end if;
64              resultado <= "0000000000000000";
65          when "1010" | "1011" => --mult
66              resultado <= rs(7 downto 0) * rd(7 downto 0);
67          when others => --J
68              resultado <= "0000000000000000";
69          end case;
70      end process;
71  end architecture;
```

COMPONENTES

MULTIPLEXADOR

- Trecho do código do multiplexador

```
--Porta multiplexador
--Bibliotecas e pacotes
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;
--Entidade
entity multiplexador is
    port(
        a, b:    in std_logic_vector(15 downto 0);
        s:       in std_logic;
        z:       out std_logic_vector(15 downto 0)
    );
end entity;

--Arquitetura
architecture Behavior of multiplexador is
begin
    z <=  a when s = '0'
    else b;
end architecture;
```

COMPONENTES

UNIDADE DE CONTROLE

- Trecho do código da UC

```
 6   --Entidade
 7   entity Uc is
 8   PORT(
 9           Clock: in std_logic;
10           OpCode    : in std_logic_vector(3 downto 0);
11           AluOP     : out std_logic_vector(3 downto 0);
12           Jump,Branch,MemRead,MemtoReg,MemWrite,RegWrite,AluSrc: out std_logic
13       );
14   end entity;
15   architecture Behavior of Uc is
16   begin
17      process(Clock,OpCode)
18      Constant Add   : Std_logic_vector(3 downto 0) := "0000";
19      Constant Addi  : Std_logic_vector(3 downto 0) := "0001";
20      Constant Sub   : Std_logic_vector(3 downto 0) := "0010";
21      Constant Subi  : Std_logic_vector(3 downto 0) := "0011";
22      Constant Lw    : Std_logic_vector(3 downto 0) := "0100";
23      Constant Sw    : Std_logic_vector(3 downto 0) := "0101";
24      Constant Li    : STD_LOGIC_VECTOR(3 downto 0) := "0110";
25      Constant Beq   : Std_logic_vector(3 downto 0) := "0111";
26      Constant If_op : std_logic_vector(3 downto 0) := "1000";
27      Constant J     : Std_logic_vector(3 downto 0) := "1001";
28      Constant Mult  : Std_logic_vector(3 downto 0) := "1010";
29      Constant Multi : Std_logic_vector(3 downto 0) := "1011";
30
31
32      begin
33         case OpCode is
34            when Add =>
35               AluOp    <= "0000";
36               Regwrite <= '1';
37               Jump     <= '0';
38               Branch   <= '0';
39               MemRead  <= '0';
40               MemToReg <= '0';
41               MemWrite <= '0';
42               Alusrc   <= '0';
43
```

COMPONENTES

MEM. RAM

- Trecho do código da memória RAM

```
1   library IEEE;
2   USE IEEE.STD_LOGIC_1164.ALL;
3   USE IEEE.numeric_std.ALL;
4
5   entity Ram is
6   port (
7       clock,WRITE_MEMORY, READ_MEMORY: in std_logic;
8       ENDERECO: IN STD_LOGIC_VECTOR (15 DOWNTO 0);
9       DATA_IN: IN STD_LOGIC_VECTOR (15 DOWNTO 0);
10      DATA_OUT: out STD_LOGIC_VECTOR (15 DOWNTO 0)
11      );
12  end entity;
13
14  architecture Behavior of ram is
15  TYPE MemList IS ARRAY (0 TO 15) OF STD_LOGIC_VECTOR(15 DOWNTO 0);
16  signal RAM_data: MemList := (others => "0000000000000000");
17  BEGIN
18      PROCESS(clock)
19      BEGIN
20          IF(rising_edge(clock)) THEN
21              IF (WRITE_MEMORY = '1') THEN
22                  RAM_data(to_integer(unsigned(ENDERECO))) <= DATA_IN;
23              END IF;
24              IF (READ_MEMORY = '1') THEN
25                  DATA_OUT <= RAM_data(to_integer(unsigned(ENDERECO)));
26              END IF;
27          END IF;
28      END PROCESS;
29  END;
```

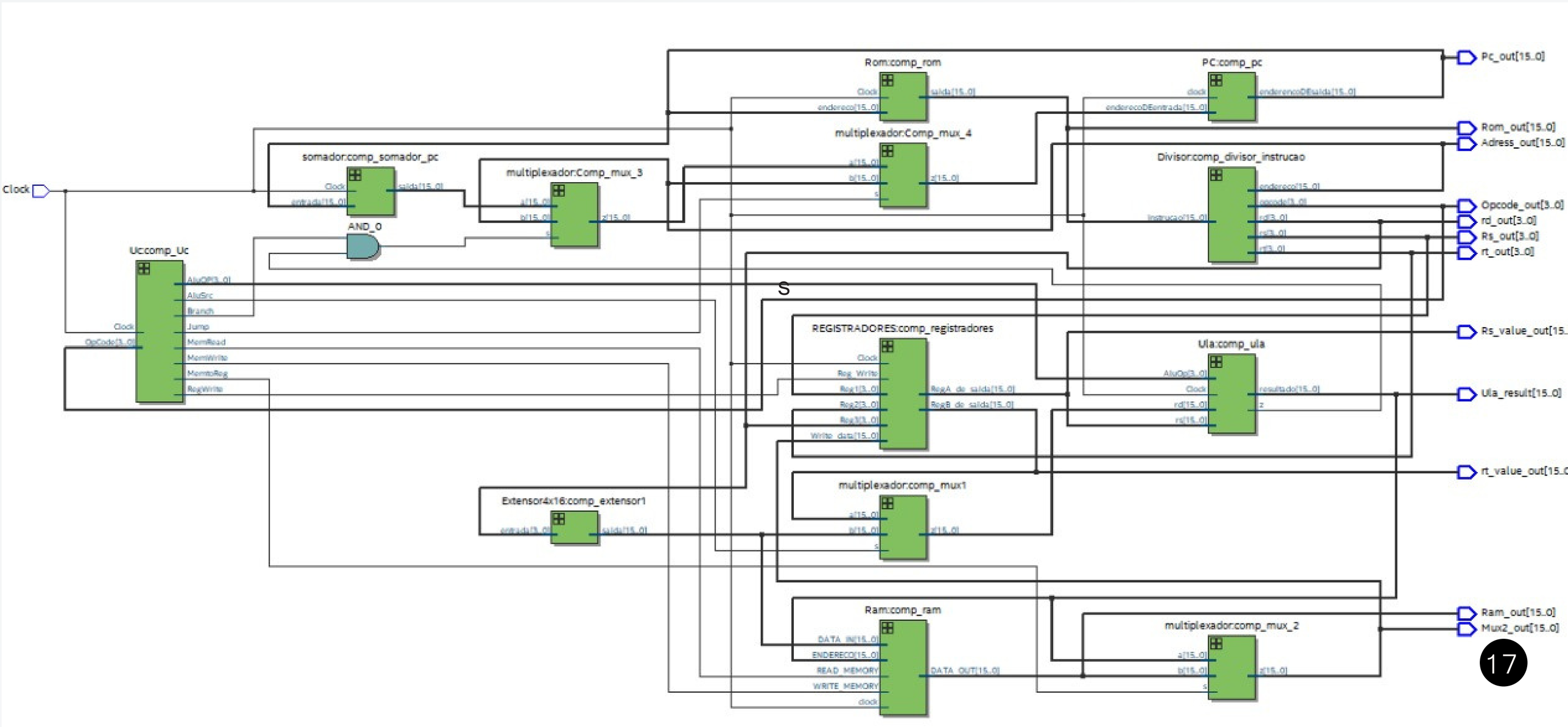
COMPONENTES

BANCO DE REGISTRADORES

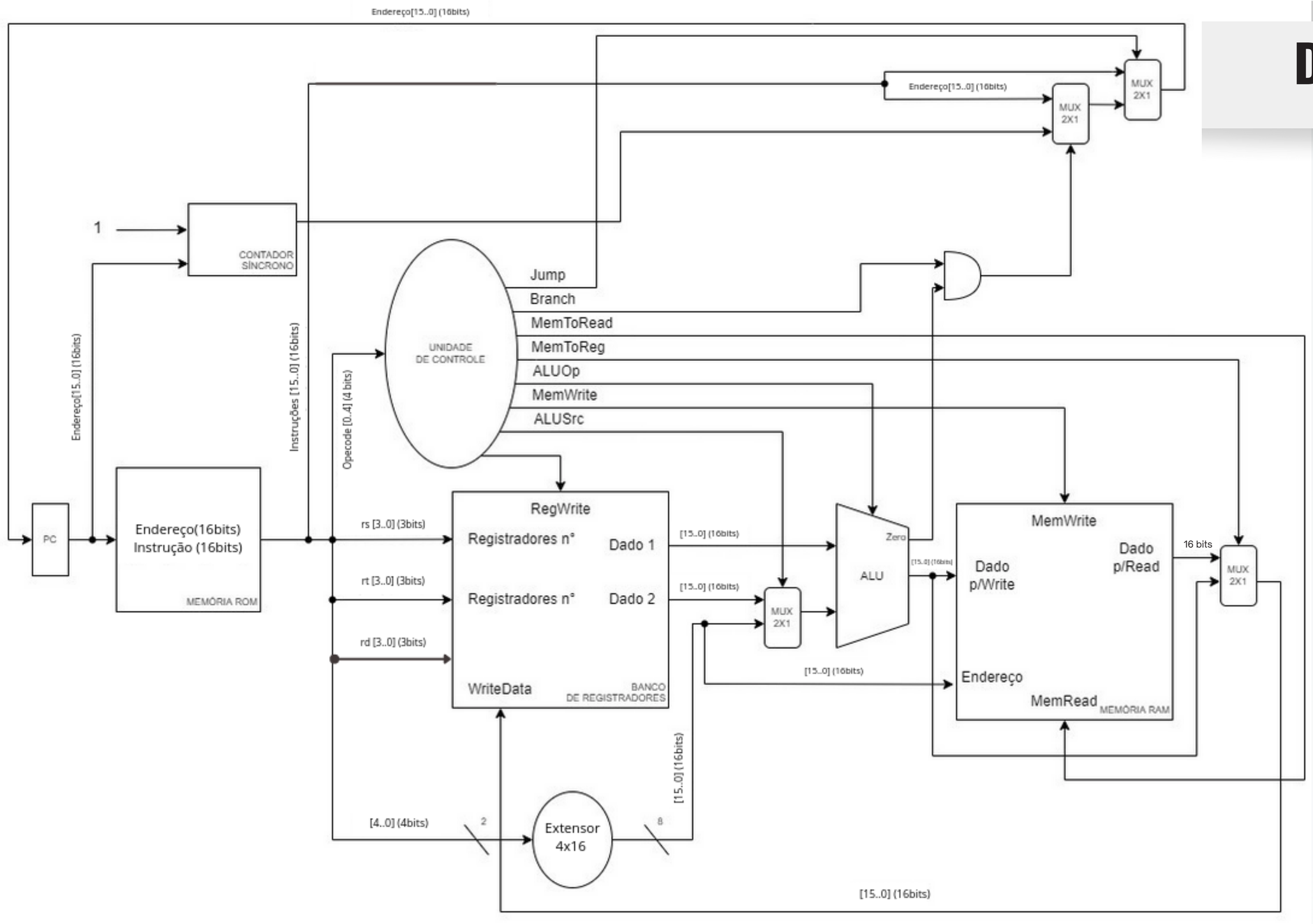
- Trecho do código do código do Banco de registradores

```
1    LIBRARY IEEE;
2   USE IEEE.STD_LOGIC_1164.ALL;
3   USE IEEE.STD_LOGIC_UNSIGNED.ALL;
4   USE IEEE.NUMERIC_STD.ALL;
5
6   Entity REGISTRADORES is
7      port(
8      Clock, Reg_Write: IN STD_LOGIC;
9      Reg1: IN STD_LOGIC_VECTOR (3 DOWNTO 0); -- nomeclatura do vetor
10     Reg2: IN STD_LOGIC_VECTOR (3 DOWNTO 0); -- nomeclatura do vetor
11     Reg3: IN STD_LOGIC_VECTOR (3 DOWNTO 0); -- nomeclatura do vetor
12     Write_data: IN STD_LOGIC_VECTOR (15 DOWNTO 0);
13     RegA_de_saida: OUT STD_LOGIC_VECTOR (15 DOWNTO 0);
14     RegB_de_saida: OUT STD_LOGIC_VECTOR (15 DOWNTO 0)
15     );
16  END REGISTRADORES;
17  ARCHITECTURE BEHAVIOR OF REGISTRADORES IS
18  TYPE REGISTRADORES IS ARRAY (0 TO 8) OF STD_LOGIC_VECTOR (15 DOWNTO 0);
19  SIGNAL MEM_REGISTRADORES: REGISTRADORES;
20  BEGIN
21     PROCESS(Clock, Reg1, Reg2)
22     BEGIN
23        IF RISING_EDGE(Clock) THEN
24           IF (Reg_Write = '1') THEN
25              MEM_REGISTRADORES(TO_INTEGER(UNSIGNED(Reg1))) <= Write_data;
26           END IF;
27        END IF;
28        RegA_de_saida <= MEM_REGISTRADORES(TO_INTEGER(UNSIGNED(Reg2)));
29        RegB_de_saida <= MEM_REGISTRADORES(TO_INTEGER(UNSIGNED(Reg3)));
30     END PROCESS;
31  END architecture;
32
```

MAPA GERAL DO INSALUBYTE



DATAPATH


Endereço[15..0] (16bits)
Endereço[15..0] (16bits)
MUX 2X1
MUX 2X1
1
CONTADOR SÍNCRONO
Jump
Branch
MemToRead
MemToReg
ALUOp
MemWrite
ALUSrc
UNIDADE DE CONTROLE
Endereço[15..0] (16bits)
Instruções [15..0] (16bits)
Opecode [0..4] (4 bits)
PC
Endereço(16bits)
Instrução (16bits)
MEMÓRIA ROM
RegWrite
rs [3..0] (3bits)
rt [3..0] (3bits)
rd [3..0] (3bits)
Registradores n°
Dado 1
Registradores n°
Dado 2
WriteData
BANCO DE REGISTRADORES
[15..0] (16bits)
[15..0] (16bits)
MUX 2X1
Zero
ALU
[15..0] (16bits)
MemWrite
Dado p/Write
Dado p/Read
Endereço
MemRead
MEMÓRIA RAM
16 bits
MUX 2X1
[15..0] (16bits)
[4..0] (4bits)
2
Extensor 4x16
8
[15..0] (16bits)
[15..0] (16bits)

TESTES

EXEMPLO DO CÓDIGO FIBONACCI

Endereço	Linguagem de Alto Nível	Binário			
		Opcode	Reg1	Reg2	Reg3
			Endereço		
		Dado			
0	**Addi** $S0 0	0001	0000	0000	0000
		0001000000000000			
1	**Sw** $S0	0101	0000	0000	0000
		0101000000000000			
2	**Addi** $S0, 1	0001	0000	0000	0001
		0001000000000001			
3	**Addi** $S1, 1	0001	0010	0001	0001
		0001000100010001			
4	**Lw** $S2, 0	0100	0010	0010	0000
		0100001000100000			
5	**Add** $S2, $S1	0000	0010	0010	0001
		0000001000100001			
6	**Add** $S1, $S0	0000	0001	0001	0000
		0000000100010000			
7	**Lw** $S0, 0	0100	0000	0000	0000
		0100000000000000			
8	**Add** $S0, $S2	0000	0000	0000	0010
		0000000000000010			
9	**J** 0100	1001	0000	0000	0100
		1001000000000100			

REFERÊNCIAS

1. https://www.fpga4student.com/2017/09/vhdl-code-for-mips-processor.html
2. https://allaboutfpga.com/vhdl-code-flipflop-d-t-jk-sr/
3. https://github.com/ed-henrique/8-bit-CPU/blob/main/CPU_EK/SOMADOR_8BITS.vhd
4. https://github.com/nataliaalmada/AOC_2GabrielENatalia_UFRR_2022/blob/main/Componentes/MemoriaRAM.vhd