Instituto Superior Técnico

Mestrado Integrado em Engenharia Electrotécnica e de Computadores

Arquitecturas Avançadas de Computadores

Descrição do processador μRisc a funcionar em *pipeline*

Guilherme Branco Teixeira	n.º 70214
Maria Margarida Dias dos Reis	n.º 73099
Nuno Miguel Rodrigues Machado	n.º 74236

Lisboa, 10 de Maio 2015

Índice

1 Introdução

Com este trabalho laboratorial pretende-se projectar um processador μRisc com funcionamento em *pipeline*. O processador possui 4 andares de *pipelining*: no primeiro andar é feito o *instruction fetch* (IF), no segundo andar é feito o *instruction decode* (ID) e o *operand fetch* (OF), no terceiro andar são executadas operações da ALU (EX) e de acesso à memória de dados (MEM) e, por fim, no quarto é feita a escrita no banco de registos, o *write back* (WB). Com o funcionamento em *pipelining* podem ocorrer dois tipos de conflitos - de dados *(data hazards)* e de controlo *(control hazards)*.

estrutura ideal do relatorio: 1:referir no geral que conflitos ha. 2:referir quais os conflitod que temos neste processador. 3:referir como os resolvemos. 4:comparar as arquitecturas que testámos. 5:entregar antes da meia-noite LooLoL

2 Conflitos associados a uma arquitectura *pipeline*

Read after Write é um conflito de dados que acontece quando uma instrução precisa de ler um valor que ainda não foi escrito na memória pois pertence a uma intrução anterior que ainda não escreveu no seu registo de destino.

Write after Read é um conflito de dados que que ocorre quando uma instrução necessita de escrever num registo quando a uma instrução anterior ainda não leu o valor desse registo.

Write after Write é um conflito quando duas operações necessitam de escrever no mesmo registo ao mesmo tempo ou numa ordem incorrecta.

Os conflitos *Write after Read* e *Write after Write* não ocorrem no nosso processador pois não existe qualquer tipo de *bypassing* entre os seus andares, mantendo sempre a ordem das instruções intacta.

2.1 Conflitos estruturais

2.2 Conflitos de dados

aqui por exemplo nos so vamos ter RAW. mas temos de explicar todos os que ha

2.3 Conflitos de controlo

3 Métodos de resolução de conflitos

Em primeiro lugar apresentam-se os métodos e técnicas de resolução dos conflitos de controlo e dados. Em segundo lugar quais as técnicas que foram de facto utilizadas.

3.1 Conflitos de dados

Conflito que surge quando uma instrução depende dos resultados de uma instrução anterior, de forma a afectar o resultado obtido pela linha de processamento.

- **Solução 1**: Bloqueio dos andares do *pipeline*, *stall*, até que os dados correctos estejam disponíveis;

- **Solução 2**: Se o dado correcto existir algures no *pipeline*, estabelece-se um *bypass* para o andar correcto, aplicando a técnica de *forwarding*;
- **Solução 3**: Escalonar/reordenar as instruções, se a ordenação for feita pelo compilador, tem-se um escalonamento estático, se for feita pelo *hardware*, escalonamento dinâmico;

3.2 Conflitos de controlo

Um conflito de controlo surge quando uma instrução de controlo condicional depende dos resultados de uma instrução anterior, de uma forma a impedir um predição correcta. Analisando as soluções apresentadas verificou-se que o escalonamento estático e dinâmico não seria a solução desejada devido a complexidade para um processador de 4 andares comparativamente às outras anteriores. Ponderou-se inicialmente a utilização de *stalls* devido à facilidade de implementação mas devido ao inconveniente de reduzir o número médio de instruções por ciclo, optou-se pela segunda solução de forma a aumentar o número médio de instruções por ciclo como também a complexidade de implementação.

nao impede a prediccao

- **Solução 1**: BTB, uma tabela que contém informação sobre os saltos de forma a prever se estes são *taken* ou *not-taken*, previsão esta que é realizada por uma BPB (composta por 1 ou 2 bits), diminuindo assim o número de ciclos desperdiçados em instruções do tipo controlo;
- **Solução 2**: *forwarding* de flags, após se obter o resultado necessário para a predição, estabelece-se um bypass para verificar a condição do salto;

4 Estrutura do processador

5 Testes de *performance*

Tabela 1: *Performance* obtida para os diversos testes com o processador demonstrado na aula.

Processador Demonstrado	Frequência [MHz]	Número de Ciclos de Execução	Número de Predicções Falhadas	Tempo de Excução [µs]
Teste #1	183,169	57	0	0,3112
Teste #2	183,169	2358	108	12,8734
Teste #3	183,169	1058	122	5,7761

Para se perceber melhor a influência que os métodos utilizados têm no desempenho do processador foram realizados vários testes para o processador projectado (Processador #1) e outros três processadores diferentes, cada processador apresenta diferentes combinações de métodos usados para corrigir os conflitos de dados e controlo, tal como se pode observar na Tabela 2.

Tabela 2: Diversas topologias do processador que foram testadas.

Processador #1	com *forwarding* de dados e BTB
Processador #2	sem *forwarding* de dados, com NOPS e com BTB
Processador #3	com *forwarding* de dados e sem BTB
Processador #4	sem *forwarding* de dados e sem BTB

Tabela 3: Resultados obtidos para o teste#1.

Teste #1	Frequência [MHz]	Número de Ciclos de Execução	Número de Predicções Falhadas	Tempo de Excução [µs]
Processador #1	211,338	57	0	0,2697
Processador #2	211,338	114	0	0,5394
Processador #3	285,608	57	0	0,1996
Processador #4	285,608	114	0	0,3991

Tabela 4: Resultados obtidos para o teste#2.

Teste #2	Frequência [MHz]	Número de Ciclos de Execução	Número de Predicções Falhadas	Tempo de Excução [µs]
Processador #1	211,338	2358	108	11,1575
Processador #2	211,338	3044	108	14,4035
Processador #3	285,608	2705	463	9,4710
Processador #4	285,608	3323	463	11,6348

Tabela 5: Resultados obtidos para o teste#3.

Teste #3	Frequência [MHz]	Número de Ciclos de Execução	Número de Predicções Falhadas	Tempo de Excução [µs]
Processador #1	211,338	1058	122	5,0062
Processador #2	211,338	1582	122	7,4856
Processador #3	285,608	1123	187	3,9320
Processador #4	285,608	1647	187	5,7666

Em análise aos resultados dos três testes realizados (Tabelas 3, 4 e 5), foi possível tirar as seguintes conclusões em relação ao *forwarding* de dados e à BTB.

5.0.0.1 *forwarding* de dados

Ao observar as caracteristicas dos quatro processadores, percebemos que é comparando os resultados entre os processadores #1 e #2 e também entre os processadores #3 e #4 que conseguimos avaliar os efeitos de usar *forwarding* de dados.

Em relação ao primeiro teste (Tabela)

referir tabela 4

5.0.0.2 BTB

6 Conclusões

7 Anexos

7.1 Código do andar de IF

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity InF is
port(
  -- input
  clk, rst              : in std_logic;
  en_if                 : in std_logic;
  reg_pc_IN             : in std_logic_vector(11 downto 0);   -- PC realimentado

  pc_branch             : in std_logic_vector(11 downto 0);   -- acrescentar
  Jump_BTB              : in std_logic_vector(11 downto 0);
  OUT_BTB               : in std_logic_vector(16 downto 0);
  act_BTB               : in std_logic;
  Conflit_DADOS         : in std_logic;
  en_Jump_FII           : in std_logic;
  en_Jump_Incond        : in std_logic;
  count_IN              : in std_logic_vector(15 downto 0);
  Jump_Incon            : in std_logic_vector(11 downto 0);
  out_ROM               : in std_logic_vector(15 downto 0);

  -- output
  addr_BTB_A            : out std_logic_vector(8 downto 0);
  addr_BTB_B            : out std_logic_vector(8 downto 0);
  IN_BTB                : out std_logic_vector(16 downto 0);
  we_BTB                : out std_logic;
  reg_PCMEM_OUT         : out std_logic_vector(11 downto 0);   -- PC + 1
  reg_pc_OUT            : out std_logic_vector(11 downto 0);   -- PC realimentado
  addr                  : out std_logic_vector(11 downto 0);   -- PC + 1
  reg_IfOut_ROM         : out std_logic_vector(15 downto 0);
  reg_IfOut_MUX_BrPred  : out std_logic;
  count_OUT             : out std_logic_vector(15 downto 0);
  reg_IfOut_PC          : out std_logic_vector(11 downto 0)    -- registo entre
    andares
  );
end InF;

architecture Behavioral of InF is

  ---------------------------------------------------------------------------
  -------------------------- Aux Signals ------------------------------------
```

```
  ---------------------------------------------------------------------------
  signal aux_pc_add_1       : std_logic_vector(11 downto 0) := (others => '0');
  signal aux_count          : std_logic_vector(15 downto 0) := (others => '0');
  signal signal_count       : std_logic := '0';
  signal aux_saida_mux      : std_logic_vector(11 downto 0) := (others => '0');
  signal aux_reg_pc         : std_logic_vector(11 downto 0) := (others => '0');
  signal aux_reg_pc_backup  : std_logic_vector(11 downto 0) := (others => '0');
  signal aux_out_ROM        : std_logic_vector(15 downto 0) := (others => '0');
  signal Aux_IN_BTB         : std_logic_vector(16 downto 0) := (others => '0');
  signal aux_pc_OUT         : std_logic_vector(11 downto 0) := (others => '0');
  ----Sinais para o BTB
  signal MSB_PC_TAG         : std_logic := '0';
  signal MUX_BrPred         : std_logic := '0';
  signal aux_we_BTB         : std_logic := '0';
  signal Addr_BTB_Act       : std_logic_vector( 8 downto 0) := (others => '0');
  signal MSB_BTB            : std_logic_vector( 2 downto 0) := (others => '0');
  signal PC_BTB             : std_logic_vector( 11 downto 0) := (others => '0');
  signal Jump_From_BTB      : std_logic_vector( 11 downto 0) := (others => '0');
  signal Prediction_Bit     : std_logic := '0';
  signal Validate_Bit       : std_logic := '0';
  signal Clean_BTB          : std_logic := '0';
  signal Insert_BTB         : std_logic := '0';
  signal jump_equal_pc      : std_logic := '0';
  signal MUX_NEXTPC         : std_logic_vector( 1 downto 0) := (others => '0');
  signal aux_erros          : std_logic_vector(11 downto 0) := (others => '0');

---------------------------------------------------------------------------
--------------------  Constantes   ----------------------------------------
---------------------------------------------------------------------------
  constant one            : std_logic_vector(11 downto 0) := "000000000001";
  constant zeros_16       : std_logic_vector(15 downto 0) := (others => '0');
  constant zeros_12       : std_logic_vector(11 downto 0) := (others => '0');

begin

--------------------------------
------     BTB    --------------
--------------------------------

  --Leitura da BTB
  addr_BTB_B      <= aux_pc_add_1(8 downto 0);--confirmar
  MSB_BTB         <= OUT_BTB(16 downto 14);
  Jump_From_BTB   <= OUT_BTB(13 downto 2);
  Prediction_Bit  <= OUT_BTB(1);
  Validate_Bit    <= OUT_BTB(0);


```

```
90  --Actualizacao da BTB
91  addr_BTB_A   <=  pc_branch(8 downto 0);
92  IN_BTB       <= pc_branch(11 downto 9)&Jump_BTB&act_BTB&'1';
93  we_BTB       <= Insert_BTB or Clean_BTB;
94
95  --Mux Not Taken='0'/ Taken='1'
96  MSB_PC_TAG     <= ((aux_pc_add_1(11) xnor MSB_BTB(2)) and (aux_pc_add_1(10) xnor
      MSB_BTB(1)) and (aux_pc_add_1(9) xnor MSB_BTB(0))) and Validate_Bit;
97
98  MUX_BrPred     <= Prediction_Bit when MSB_PC_TAG = '1' else '0';
99
100 PC_BTB         <=  Jump_From_BTB when MUX_BrPred = '1' else
101                    reg_pc_IN + one;
102
103 MUX_NEXTPC(1) <=  Conflit_DADOS AND ( (NOT(en_Jump_Incond) AND NOT(jump_equal_pc)
      AND ((Act_BTB AND NOT(en_Jump_FII)) OR en_Jump_FII)) OR (jump_equal_pc AND NOT(
      Act_BTB) AND NOT(en_Jump_FII) AND NOT(en_Jump_Incond)) );
104 MUX_NEXTPC(0) <= Conflit_DADOS AND (en_Jump_Incond OR (jump_equal_pc AND NOT(Act_BTB
      ) AND NOT(en_Jump_FII) AND NOT(en_Jump_Incond)));
105
106 aux_saida_mux <= PC_BTB              when MUX_NEXTPC = "00" else
107                  Jump_Incon          when MUX_NEXTPC = "01" else
108                  Jump_BTB            when MUX_NEXTPC = "10" else
109                  aux_reg_pc_backup;
110
111 --aux_saida_mux <= reg_pc_IN ;
112
113 ---------------------------------------------------------------------------
114 ----------------------- Registo PC  ---------------------------------------
115 ---------------------------------------------------------------------------
116   process (clk, rst,MUX_BrPred)
117     begin
118       if clk'event and clk = '1' then
119         if rst = '1' then
120           aux_reg_pc <= zeros_12;
121         elsif MUX_BrPred = '1' then
122           aux_reg_pc_backup <= reg_pc_IN + one;
123           aux_reg_pc       <= aux_saida_mux;
124         else
125           aux_reg_pc       <= aux_saida_mux;
126         end if;
127       end if;
128   end process;
129
130   process (clk, rst,count_IN)
131     begin
132       if clk'event and clk = '1' then
```

```
133         if rst = '1' then
134           aux_count<= zeros_16;
135         elsif signal_count = '0' then
136           aux_count <= count_IN + one;
137         end if;
138       end if;
139   end process;
140 
141   aux_pc_add_1 <= aux_reg_pc;
142 
143   count_OUT <= aux_count;
144 
145   --Controlo de saltos em relacao a BTB
146 
147   jump_equal_pc <= Conflit_DADOS when Jump_BTB = aux_pc_add_1 else '0';
148 
149   -- CASOS POSSIVEIS
150   -- jump_equal_pc = 0 e act_BTB = 1 tem que carregar o salto e actualizar btb
151   -- jump_equal_pc = 1 e act_BTB = 0 tem que carregar o pc antigo e limpar btb
152 
153   Clean_BTB   <= jump_equal_pc AND Not(act_BTB) AND NOT(en_Jump_FII) AND NOT(
      en_Jump_Incond);
154   Insert_BTB  <= (NOT(jump_equal_pc) AND act_BTB AND NOT(en_Jump_FII) AND NOT(
      en_Jump_Incond)) OR (NOT(jump_equal_pc) AND en_Jump_FII AND NOT(en_Jump_Incond));
155 
156 
157   signal_count <= '1' when out_ROM = x"2fff" else '0' ;
158 ---------------------------------------------------------------------------
159 --------------------- Registo de Instrucoes (IR) --------------------------
160 ---------------------------------------------------------------------------
161   process (clk, rst)
162     begin
163       if clk'event and clk = '1' then
164         if rst = '1' then
165           aux_out_ROM <= zeros_16;
166           aux_pc_OUT  <= zeros_12;
167         elsif en_if = '1' then
168           if (Conflit_DADOS = '1') and(Clean_BTB = '1' or Insert_BTB = '1' or
      en_Jump_Incond = '1')  then
169             aux_out_ROM <= x"0000";
170             aux_erros <= aux_erros + one;
171             aux_pc_OUT  <= aux_pc_add_1;
172           else
173             aux_out_ROM <= out_ROM;
174             aux_pc_OUT  <= aux_pc_add_1 + one;    --Modificado em relacao ao
      apresentado no Laboratorio, foi acrescentado + one;
175           end if;
```

```
        end if;
      end if;
  end process;

-------------------------------------------------------------------------
------------------------------ Exit -------------------------------------
-------------------------------------------------------------------------
  reg_PCMEM_OUT    <= aux_reg_pc;
  addr             <= aux_reg_pc;
  reg_pc_OUT       <= aux_pc_add_1;
  reg_IfOut_ROM    <= aux_out_ROM;
  reg_IfOut_PC     <= aux_pc_OUT;

end Behavioral;
```

7.2 Código do andar de ID e OF

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity IDeOF is
  port(
    -- input
    clk, rst         : in std_logic;
    reg_IfOut_PC       : in std_logic_vector(11 downto 0);   -- registo entre andares
    reg_IfOut_ROM        : in std_logic_vector(15 downto 0);
    FLAGS_IN           : in std_logic_vector(3 downto 0);
    Forw_FLAGS_IN      : in std_logic_vector(3 downto 0);
    ADD_RWC_EXMEM      : in std_logic_vector(2 downto 0);
    ADD_RWC_WB         : in std_logic_vector(2 downto 0);
    ovWE_EXMEM         : in std_logic;
    ovWE_WB          : in std_logic;
    en_idOF          : in std_logic;
    R0             : in std_logic_vector(15 downto 0);
    R1             : in std_logic_vector(15 downto 0);
      R2             : in std_logic_vector(15 downto 0);
    R3             : in std_logic_vector(15 downto 0);
    R4             : in std_logic_vector(15 downto 0);
    R5             : in std_logic_vector(15 downto 0);
    R6             : in std_logic_vector(15 downto 0);
    R7             : in std_logic_vector(15 downto 0);
    --Forwarding
    Forw_EXMEN           : in std_logic_vector(15 downto 0);
    Forw_WB            : in std_logic_vector(15 downto 0);

    -- output
```

```
    en_Jump_FII          : out std_logic;
    en_Jump_Incond         : out std_logic;
    Jump_Incon           : out std_logic_vector(11 downto 0);
    pc_branch          : out std_logic_vector(11 downto 0);    -- acrescentar
    Jump_BTB             : out std_logic_vector(11 downto 0);
    act_BTB            : out std_logic;
    Conflit_ON           : out std_logic;
    Conflit_DADOS        : out std_logic;
    pc_add_jump          : out std_logic_vector(11 downto 0);
    reg_IDOF_OUT_WERAM     : out std_logic;
    reg_IDOF_OUT_ALUOPER     : out std_logic_vector(4 downto 0);
    reg_IDOF_OUT_bit15     : out std_logic;
    reg_IDOF_OUT_bit14     : out std_logic;
    reg_IDOF_OUT_OperA     : out std_logic_vector(15 downto 0);
    reg_IDOF_OUT_OperB     : out std_logic_vector(15 downto 0);
    reg_IDOF_OUT_ALUvsMEM : out std_logic;
    reg_IDOF_OUT_ovWE      : out std_logic;
    reg_IDOF_OUT_AddRWC    : out std_logic_vector(2 downto 0);
    reg_IDOF_OUT_PCadd1    : out std_logic_vector(11 downto 0);
    reg_IDOF_OUT_SelMuxWB : out std_logic_vector(1 downto 0);
    reg_IDOF_OUT_MuxConst : out std_logic_vector(15 downto 0)
  );
end IDeOF;

  architecture Behavioral of IDeOF is

-------------------------------------------------------------------------
--------------------------- Aux Signals ---------------------------------
-------------------------------------------------------------------------
signal inst_IN         : std_logic_vector(15 downto 0) := (others => '0');
signal aux_ADD_RWC       : std_logic_vector(2 downto 0) := (others => '0');
signal aux_ADD_RA         : std_logic_vector(2 downto 0) := (others => '0');
signal aux_ADD_RB         : std_logic_vector(2 downto 0) := (others => '0');
signal aux_ADD_RA_C      : std_logic_vector(2 downto 0) := (others => '0');
signal aux_ALU_OPER      : std_logic_vector(4 downto 0) := (others => '0');
signal aux_CONS_FI_11B     : std_logic_vector(10 downto 0) := (others => '0');
signal aux_CONS_FII_8B     : std_logic_vector(7 downto 0) := (others => '0');
signal aux_active_FLAGTEST  : std_logic := '0';
signal aux_FLAGMUX       : std_logic := '0';
signal aux_FLAGMUX_C_For   : std_logic := '0';
signal aux_FLAGMUX_S_For   : std_logic := '0';
signal aux_TRANS_OP      : std_logic_vector(1 downto 0) := (others => '0');
signal aux_TRANS_FI_DES    : std_logic_vector(11 downto 0) := (others => '0');
signal aux_TRANS_FII_DES   : std_logic_vector(11 downto 0) := (others => '0');
signal aux_TRANS_DES        : std_logic_vector(11 downto 0) := (others => '0');
signal aux_TRANS_FIII_R      : std_logic := '0';
signal aux_JUMPS_active    : std_logic := '0';
```

```
signal aux_JUMPS_MUX_WB   : std_logic := '0';
signal aux_pc_add_one   : std_logic_vector(11 downto 0) := (others => '0');
signal aux_TEST_NOP     : std_logic := '0';
signal aux_FLAGTEST       : std_logic := '0';
signal aux_FLAGTEST_MUXPC   : std_logic := '0';
signal JUMP_MUXWB_OUT   : std_logic := '0';
signal ALU_vs_MEM       : std_logic := '0';
signal Aux_Conflit_DADOS  : std_logic := '0';
signal RA_C           : std_logic_vector(15 downto 0) := (others => '0');
signal RB             : std_logic_vector(15 downto 0) := (others => '0');
signal oper_A         : std_logic_vector(15 downto 0):= (others => '0');  -- operando
    A para a ALU
signal oper_B         : std_logic_vector(15 downto 0):= (others => '0');  -- operando
    B para a ALU
signal const11          : std_logic_vector(15 downto 0):= (others => '0');  --
    operando B para a ALU
signal lcl            : std_logic_vector(15 downto 0):= (others => '0');  -- operando
    B para a ALU
signal lch            : std_logic_vector(15 downto 0):= (others => '0');  -- operando
    B para a ALU
signal select_mux_constantes: std_logic_vector(1 downto 0):= (others => '0');
signal out_mux_constantes   : std_logic_vector(15 downto 0):= (others => '0');  --
    operando para carregamento de constantes
signal bit14            : std_logic := '0';         -- sinal de selecaao para MUX
    entre operacao da ALU e operacao de carregamento de constantes
signal bit15            : std_logic := '0';
signal WE_RAM           : std_logic := '0';
signal TRANS_FI_COND_IN     : std_logic_vector(3 downto 0) := (others => '0');
signal pc_add1        : std_logic_vector(11 downto 0):= (others => '0');
signal mux_jump_cond    : std_logic_vector(1 downto 0) := (others => '0');
signal aux_saida_mux    : std_logic_vector(11 downto 0):= (others => '0');
signal ALU_e_MEM      : std_logic := '0';
signal soMEM        : std_logic := '0';
signal soALU        : std_logic := '0';
signal loadMEM        : std_logic := '0';
signal controlo       : std_logic := '0';
signal controloJump     : std_logic := '0';
signal ovWE           : std_logic := '0';
signal aux_sel_bit1     : std_logic := '0';
signal aux_sel_bit0     : std_logic := '0';
signal sel_mux_WB       : std_logic_vector(1 downto 0) := (others => '0');
signal aux_reg_IDOF_OUT_WERAM   : std_logic := '0' ;
signal aux_reg_IDOF_OUT_ALUOPER   : std_logic_vector(4 downto 0) := (others => '0');
signal aux_reg_IDOF_OUT_bit15   : std_logic :='0';
signal aux_reg_IDOF_OUT_bit14   : std_logic :='0';
signal aux_reg_IDOF_OUT_OperA   : std_logic_vector(15 downto 0) := (others => '0');
signal aux_reg_IDOF_OUT_OperB   : std_logic_vector(15 downto 0) := (others => '0');
```

```
signal aux_reg_IDOF_OUT_ALUvsMEM  : std_logic :='0';
signal aux_reg_IDOF_OUT_ovWE    : std_logic :='0';
signal aux_reg_IDOF_OUT_AddRWC    : std_logic_vector(2 downto 0) := (others => '0');
signal aux_reg_IDOF_OUT_PCadd1    : std_logic_vector(11 downto 0) := (others => '0');
signal aux_reg_IDOF_OUT_SelMuxWB  : std_logic_vector(1 downto 0) := (others => '0');
signal aux_reg_IDOF_OUT_MuxConst  : std_logic_vector(15 downto 0) := (others => '0');
--Conflito
signal RA_C_EXMEM_CONFLITO : std_logic :='0';
signal RA_C_WB_CONFLITO : std_logic :='0';
signal RB_EXMEM_CONFLITO : std_logic :='0';
signal RB_WB_CONFLITO : std_logic :='0';
signal aux_Conflit_ON : std_logic :='0';
signal Conflit_EXMEN_RA_C_ON : std_logic:='0';
signal Conflit_WB_RA_C_ON : std_logic:='0';
signal Conflit_EXMEN_RB_ON : std_logic:='0';
signal Conflit_WB_RB_ON : std_logic:='0';
signal mux_RA       : std_logic_vector(1 downto 0) := (others => '0');
signal mux_RB     : std_logic_vector(1 downto 0) := (others => '0');
signal aux_oper_A   : std_logic_vector(15 downto 0) := (others => '0');



--------------------------------------------------------------------------
--------------------  Constantes   ---------------------------------------
--------------------------------------------------------------------------
constant one          : std_logic_vector(11 downto 0) :="000000000001" ;
constant zero_12        : std_logic_vector(11 downto 0) := (others => '0');
constant zero_16        : std_logic_vector(15 downto 0) := (others => '0');

begin

inst_IN  <=  reg_IfOut_ROM;
pc_add1  <=  reg_IfOut_PC; --Modificado em relacao ao apresentado no Laboratorio, foir
    retirado + one;

aux_ADD_RWC <= inst_IN(13 downto 11);
aux_ADD_RA  <= inst_IN(5 downto 3);
aux_ADD_RB  <= inst_IN(2 downto 0);

aux_ADD_RA_C <= aux_ADD_RWC when (inst_IN(15) and inst_IN(14)) = '1' else
       aux_ADD_RA;

--------------------------------------------------------------------------
------------------------ Id conflito -------------------------------------
--------------------------------------------------------------------------

RA_C_EXMEM_CONFLITO   <=  ((ADD_RWC_EXMEM(2) XNOR aux_ADD_RA_C(2)) and (ADD_RWC_EXMEM
```

```
    (1) XNOR aux_ADD_RA_C(1)) and (ADD_RWC_EXMEM(0) XNOR aux_ADD_RA_C(0)))  and
    ovWE_EXMEM;
RA_C_WB_CONFLITO      <=  ((ADD_RWC_WB(2) XNOR aux_ADD_RA_C(2)) and (ADD_RWC_WB(1)
    XNOR aux_ADD_RA_C(1)) and (ADD_RWC_WB(0) XNOR aux_ADD_RA_C(0)))  and ovWE_WB;

RB_EXMEM_CONFLITO   <=  ((ADD_RWC_EXMEM(2) XNOR aux_ADD_RB(2)) and (ADD_RWC_EXMEM(1)
    XNOR aux_ADD_RB(1)) and (ADD_RWC_EXMEM(0) XNOR aux_ADD_RB(0)))  and ovWE_EXMEM;
RB_WB_CONFLITO      <=  ((ADD_RWC_WB(2) XNOR aux_ADD_RB(2)) and (ADD_RWC_WB(1) XNOR
    aux_ADD_RB(1)) and (ADD_RWC_WB(0) XNOR aux_ADD_RB(0)))  and ovWE_WB;


Conflit_EXMEN_RA_C_ON   <= RA_C_EXMEM_CONFLITO and inst_IN(15);
Conflit_WB_RA_C_ON    <= RA_C_WB_CONFLITO   and inst_IN(15);
Conflit_EXMEN_RB_ON   <= (RB_EXMEM_CONFLITO  and inst_IN(15)) OR (RB_EXMEM_CONFLITO
    AND NOT(inst_IN(15)) AND NOT(inst_IN(14)) AND inst_IN(13) AND inst_IN(12));
Conflit_WB_RB_ON      <= (RB_WB_CONFLITO  and inst_IN(15)) OR (RB_EXMEM_CONFLITO AND
    NOT(inst_IN(15)) AND NOT(inst_IN(14)) AND inst_IN(13) AND inst_IN(12));

-------------------------------------------------------------------------
------ Conjuntos de instruccoes ------------------------------------------
------ Inst_IN(15:14)           ------------------------------------------
------ 0 0 => Transferencia de Controlo    -------------------------------
------ 0 1 => Constantes Formato I        -------------------------------
------ 1 0 => Instrucoes para ALU/Memoria  -------------------------------
------ 1 1 => Constante Formato II        -------------------------------
-------------------------------------------------------------------------

aux_active_FLAGTEST <= inst_IN(15) or inst_IN(14); --- Activa a FLAGTESTE
aux_TEST_NOP <= inst_IN(15) or inst_IN(14) or inst_IN(13) or inst_IN(12) or
              inst_IN(11) or inst_IN(10) or inst_IN(9) or inst_IN(8) or
              inst_IN(7) or inst_IN(6) or inst_IN(5) or inst_IN(4) or
              inst_IN(3) or inst_IN(2) or inst_IN(1) or inst_IN(0);
-------------------------------------------------------------------------
-------- 0 0 -> Transferencia de Controlo -----------------------------
-------- exitsem 3 formatos -------------------------------------------
-------------------------------------------------------------------------
aux_TRANS_OP    <= inst_IN(13 downto 12);

-------- 0 0/ 0 1 -> Formato I condicional ----------------------------

aux_TRANS_FI_DES  <= (11 downto 8 => inst_IN(7)) & inst_IN(7 downto 0);
TRANS_FI_COND_IN  <= inst_IN(11 downto 8 );

-------- 1 0 -> Formato II incondicional ------------------------------
aux_TRANS_FII_DES <= inst_IN(11 downto 0);

-------- 1 1 -> Formato III jumps -------------------------------------
```

```
205
206 aux_JUMPS_active  <= not(aux_active_FLAGTEST) and inst_IN(13) and inst_IN(12); ----- TESTE de activacao do FIII, escolhe o mux2:1
207 aux_JUMPS_MUX_WB  <= not(inst_IN(11)) and aux_JUMPS_active;
208
209 ---------------------------------------------------------------------------
210 -------- 0 1 -> Constantes  Formato I   ----------------------------------
211 ---------------------------------------------------------------------------
212
213 aux_CONS_FI_11B   <=  inst_IN(10 downto 0 );
214
215
216 ---------------------------------------------------------------------------
217 -------- 1 0 -> Instruccoes para a ALU/Memoria ----------------------------
218 ---------------------------------------------------------------------------
219
220 aux_ALU_OPER    <=  inst_IN(10 downto 6);
221
222 ---------------------------------------------------------------------------
223 -------- 1 1 -> Constantes  Formato II  ----------------------------------
224 ---------------------------------------------------------------------------
225 aux_CONS_FII_8B <=  inst_IN(7 downto 0);
226
227
228 --  ------------------------------------------------------------------------------------------------

229 ---------------------------------- TESTE FLAGS  ------------------------------------------------
230 --  ------------------------------------------------------------------------------------------------

231 aux_FLAGMUX_S_For  <=FLAGS_IN(3)          when TRANS_FI_COND_IN = "0101" else
232               FLAGS_IN(2)           when TRANS_FI_COND_IN = "0100" else
233               FLAGS_IN(1)           when TRANS_FI_COND_IN = "0110" else
234               FLAGS_IN(0)           when TRANS_FI_COND_IN = "0011" else
235               '1'                  when TRANS_FI_COND_IN = "0000" else
236               FLAGS_IN(3) or FLAGS_IN(2) when TRANS_FI_COND_IN = "0111" else
237               '0';
238
239 aux_FLAGMUX_C_For  <=Forw_FLAGS_IN(3)               when TRANS_FI_COND_IN = "0101" else
240               Forw_FLAGS_IN(2)               when TRANS_FI_COND_IN = "0100" else
241               Forw_FLAGS_IN(1)              when TRANS_FI_COND_IN = "0110" else
242               Forw_FLAGS_IN(0)              when TRANS_FI_COND_IN = "0011" else
243               '1'                          when TRANS_FI_COND_IN = "0000" else
244               Forw_FLAGS_IN(3) or Forw_FLAGS_IN(2) when TRANS_FI_COND_IN = "0111" else
```

```
245                '0';
246
247 aux_FLAGMUX <=  aux_FLAGMUX_C_For when  not(aux_active_FLAGTEST) = '1' else
    aux_FLAGMUX_S_For;
248
249
250 aux_FLAGTEST    <= (not(aux_TRANS_OP(1)) and aux_TRANS_OP(0) and aux_FLAGMUX) or (not(
    aux_TRANS_OP(1)) and not(aux_TRANS_OP(0))and not(aux_FLAGMUX));
251
252 aux_FLAGTEST_MUXPC <=  not(aux_active_FLAGTEST) and aux_FLAGTEST;
253
254 ------------------------------------------------------------------------
255 ------------------------- Operand Fetch  -------------------------------
256 ------------------------------------------------------------------------
257
258 RA_C <=  R0 when aux_ADD_RA_C = "000" else
259       R1 when aux_ADD_RA_C = "001" else
260       R2 when aux_ADD_RA_C = "010" else
261       R3 when aux_ADD_RA_C = "011" else
262       R4 when aux_ADD_RA_C = "100" else
263       R5 when aux_ADD_RA_C = "101" else
264       R6 when aux_ADD_RA_C = "110" else
265       R7;
266
267
268 RB <=   R0 when aux_ADD_RB = "000" else
269     R1 when aux_ADD_RB = "001" else
270     R2 when aux_ADD_RB = "010" else
271     R3 when aux_ADD_RB = "011" else
272     R4 when aux_ADD_RB = "100" else
273     R5 when aux_ADD_RB = "101" else
274     R6 when aux_ADD_RB = "110" else
275     R7;
276 -- operando A da ALU
277 mux_RA <= (Conflit_WB_RA_C_ON&Conflit_EXMEN_RA_C_ON);
278 aux_oper_A <= Forw_EXMEN  when mux_RA = "01" else
279           Forw_EXMEN  when mux_RA = "11" else
280           Forw_WB     when mux_RA = "10" else
281           RA_C;
282 oper_A <= aux_oper_A;
283 -- operando B da ALU
284 mux_RB <= Conflit_WB_RB_ON&Conflit_EXMEN_RB_ON;
285 oper_B <= Forw_EXMEN  when mux_RB = "01" else
286         Forw_EXMEN  when  mux_RB = "11" else
287         Forw_WB     when mux_RB = "10"   else
288         RB ;
289
```

```
290
291 const11 <= (15 downto 11 => aux_CONS_FI_11B(10)) & aux_CONS_FI_11B; -- loadlit c
292 lcl <=  aux_oper_A(15 downto 8) & aux_CONS_FII_8B;          -- lcl c
293 lch <=  aux_CONS_FII_8B & aux_oper_A(7 downto 0);           -- lch c
294
295 select_mux_constantes <= inst_IN(15) & inst_IN(10);
296
297 out_mux_constantes <=     const11   when select_mux_constantes = "00" else
298                     const11   when select_mux_constantes = "01" else
299                       lcl     when select_mux_constantes = "10" else
300                     lch;
301
302 ALU_vs_MEM <= (aux_ALU_OPER(1) and not(aux_ALU_OPER(2))) and (aux_ALU_OPER(3) and not(
    aux_ALU_OPER(4)));
303
304 WE_RAM <= (inst_IN(15) and not(inst_IN(14))) and (ALU_vs_MEM and inst_IN(6));
305
306
307
308 bit15 <= inst_IN(15);
309 bit14 <=  inst_IN(14);
310
311 ---------------------------------------------------------------------------
312 -------------Saltos Condicionais e incodicionais-----------------------
313 ---------------------------------------------------------------------------
314 Aux_Conflit_DADOS <= not(aux_active_FLAGTEST) AND aux_TEST_NOP;
315
316 act_BTB     <= aux_FLAGTEST_MUXPC;
317 en_Jump_FII   <= aux_TRANS_OP(1) and not(aux_TRANS_OP(0));
318 en_Jump_Incond  <= aux_JUMPS_active; --acrescentar e fazer logica
319
320 pc_branch   <= reg_IfOut_PC - one; --Modificado em relacao ao apresentado no
    Laboratorio, foi acrescentado -one;
321 JUMP_MUXWB_OUT <= aux_JUMPS_MUX_WB;
322
323
324 -------------Escolha da constante do destino----------------------------
325 Jump_Incon   <= oper_B(11 downto 0);
326
327 Jump_BTB      <=  aux_TRANS_FII_DES + pc_add1 when aux_TRANS_OP = "10" else
328             aux_TRANS_FI_DES  + pc_add1;
329
330 ---------------------------------------------------------------------------
331 -------------------Andar WB-- Wirte enable do registo de saida-------------
332 ---------------------------------------------------------------------------
333
334 ALU_e_MEM <= (inst_IN(15) and (not(inst_IN(14))));
```

```
335
336 soMEM <= ALU_e_MEM and ALU_vs_MEM;
337
338 soALU <= ALU_e_MEM and not(ALU_vs_MEM);
339
340 loadMEM <= soMEM and not(inst_IN(6));
341
342 controlo <= inst_IN(14) nor inst_IN(15);
343
344 controloJump <= controlo and JUMP_MUXWB_OUT;
345
346 ovWE <= (soALU or loadMEM) or (controloJump or inst_IN(14));
347
348
349 aux_sel_bit1 <= JUMP_MUXWB_OUT or inst_IN(14);
350         -- JUMP_MUXWB_OUT or inst_IN(14)
351
352 aux_sel_bit0 <= JUMP_MUXWB_OUT or (not(Inst_IN(14)) and ALU_vs_MEM) ;
353         -- JUMP_MUXWB_OUT or (not(inst_IN(14)) and ALU_vs_MEM)
354
355 sel_mux_WB <= aux_sel_bit1&aux_sel_bit0;
356 ---------------------------------------------------------------------------
357 ------------------------------ Exit -------------------------------------
358 ---------------------------------------------------------------------------
359
360 --------------- registo de saida do segundo andar: ID e OF ---------------
361 process (clk, rst)
362   begin
363     if clk'event and clk = '1' then
364       if rst = '1'  then
365         aux_reg_IDOF_OUT_bit15    <= '0'; --need
366         aux_reg_IDOF_OUT_bit14    <= '0';
367         aux_reg_IDOF_OUT_WERAM    <= '0';-- need
368         aux_reg_IDOF_OUT_ALUOPER  <= "00000"; --need
369         aux_reg_IDOF_OUT_ALUvsMEM <= '0';--need
370         aux_reg_IDOF_OUT_PCadd1   <= zero_12;--need
371         aux_reg_IDOF_OUT_SelMuxWB   <= "00"; --add
372         aux_reg_IDOF_OUT_AddRWC   <= "000"; -- need
373         aux_reg_IDOF_OUT_OperA    <=  zero_16; --need
374         aux_reg_IDOF_OUT_OperB    <= zero_16; --need
375         aux_reg_IDOF_OUT_MuxConst <= zero_16;
376         aux_reg_IDOF_OUT_ovWE       <= '0';
377       elsif en_idOF = '1' then
378           aux_reg_IDOF_OUT_bit15    <= bit15; --need
379           aux_reg_IDOF_OUT_bit14    <= bit14;
380           aux_reg_IDOF_OUT_WERAM    <= WE_RAM;-- need
381           aux_reg_IDOF_OUT_ALUOPER  <= aux_ALU_OPER; --need
```

```
382           aux_reg_IDOF_OUT_ALUvsMEM <= ALU_vs_MEM;--need
383           aux_reg_IDOF_OUT_PCadd1   <= pc_add1;--need
384           aux_reg_IDOF_OUT_SelMuxWB   <= sel_mux_WB; --add
385           aux_reg_IDOF_OUT_AddRWC   <= aux_ADD_RWC; -- need
386           aux_reg_IDOF_OUT_OperA    <=  oper_A; --need
387           aux_reg_IDOF_OUT_OperB    <= oper_B; --need
388           aux_reg_IDOF_OUT_MuxConst <= out_mux_constantes;
389           aux_reg_IDOF_OUT_ovWE       <= ovWE;
390       end if;
391     end if;
392 end process;
393 
394 reg_IDOF_OUT_bit15    <= aux_reg_IDOF_OUT_bit15; --need
395 reg_IDOF_OUT_bit14    <= aux_reg_IDOF_OUT_bit14;
396 reg_IDOF_OUT_WERAM    <= aux_reg_IDOF_OUT_WERAM;-- need
397 reg_IDOF_OUT_ALUOPER    <= aux_reg_IDOF_OUT_ALUOPER; --need
398 reg_IDOF_OUT_ALUvsMEM <= aux_reg_IDOF_OUT_ALUvsMEM;--need
399 reg_IDOF_OUT_PCadd1     <= aux_reg_IDOF_OUT_PCadd1;--need
400 reg_IDOF_OUT_SelMuxWB   <= aux_reg_IDOF_OUT_SelMuxWB; --add
401 reg_IDOF_OUT_AddRWC     <= aux_reg_IDOF_OUT_AddRWC; -- need
402 reg_IDOF_OUT_OperA    <= aux_reg_IDOF_OUT_OperA; --need
403 reg_IDOF_OUT_OperB    <= aux_reg_IDOF_OUT_OperB; --need
404 reg_IDOF_OUT_MuxConst <= aux_reg_IDOF_OUT_MuxConst;
405 reg_IDOF_OUT_ovWE     <= aux_reg_IDOF_OUT_ovWE;
406 Conflit_DADOS         <=  Aux_Conflit_DADOS;
407 
408 end Behavioral;
```

7.3 Código do andar de EX e MEM

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.STD_LOGIC_ARITH.ALL;
4  use IEEE.STD_LOGIC_UNSIGNED.ALL;
5  
6  entity EXeMEM is
7    port(
8      -- input
9      clk, rst           : in std_logic;
10     reg_IDOF_OUT_WERAM      : in std_logic;
11     reg_IDOF_OUT_ALUOPER    : in std_logic_vector(4 downto 0);
12     reg_IDOF_OUT_bit15      : in std_logic;
13     reg_IDOF_OUT_bit14      : in std_logic;
14     reg_IDOF_OUT_OperA      : in std_logic_vector(15 downto 0);
15     reg_IDOF_OUT_OperB      : in std_logic_vector(15 downto 0);
16     reg_IDOF_OUT_ALUvsMEM   : in std_logic;
17     reg_IDOF_OUT_ovWE     : in std_logic;
18     reg_IDOF_OUT_AddRWC     : in std_logic_vector(2 downto 0);
```

```
    reg_IDOF_OUT_PCadd1     : in std_logic_vector(11 downto 0);
    reg_IDOF_OUT_SelMuxWB   : in std_logic_vector(1 downto 0);
    reg_IDOF_OUT_MuxConst   : in std_logic_vector(15 downto 0);
    FLAGS_IN              : in std_logic_vector(3 downto 0);
    out_RAM             : in std_logic_vector(15 downto 0);
    en_EX               : in std_logic;

    -- output
    reg_EXMEM_OUT_PCadd1    : out std_logic_vector(11 downto 0);
    reg_EXMEM_OUT_AddRWC    : out std_logic_vector(2 downto 0);
    reg_EXMEM_OUT_OutALU    : out std_logic_vector(15 downto 0);
    reg_EXMEM_OUT_OutMEM    : out std_logic_vector(15 downto 0);
    reg_EXMEM_OUT_MuxConst    : out std_logic_vector(15 downto 0);
    reg_EXMEM_OUT_ovWE      : out std_logic;
    reg_EXMEM_OUT_SelMuxWB    : out std_logic_vector(1 downto 0);

    ADD_RWC_EXMEM       : out std_logic_vector(2 downto 0);
    ovWE_EXMEM          : out std_logic;

    out_ADD_MEM         : out std_logic_vector(11 downto 0);    -- para enderecar a
    RAM
    out_WE_MEM          : out std_logic;              -- para controlar o WE da RAM
    FLAGS_OUT         : out std_logic_vector(3 downto 0);
    Forw_FLAGSTEST_OUT    : out std_logic_vector(3 downto 0);
    Forw_EXMEN          : out std_logic_vector(15 downto 0);
    FLAGSTEST_OUT       : out std_logic_vector(3 downto 0);
    in_RAM            : out std_logic_vector(15 downto 0)

  );
end EXeMEM;

architecture Behavioral of EXeMEM is

-------------------------------------------------------------------------
--------------------------- Aux Signals ---------------------------------
-------------------------------------------------------------------------
signal out_ALU          : std_logic_vector(15 downto 0) := (others => '0'); -- saida
    da ALU
signal out_MEM          : std_logic_vector(15 downto 0) := (others => '0'); -- saida
    da memoria
signal aux_FLAGS_ARI      : std_logic_vector(3 downto 0) := (others => '0');  -- Z,N,C
    ,O
signal aux_FLAGS_SHIFT      : std_logic_vector(2 downto 0) := (others => '0');  -- Z,N
    ,C
signal aux_FLAGS_LOG      : std_logic_vector(1 downto 0) := (others => '0');  -- Z,N
signal aux_FLAGS        : std_logic_vector(3 downto 0) := (others => '0');  -- Z,N,C,O
signal aux_MSR_FLAGS      : std_logic_vector(3 downto 0) := (others => '0');
```

```
signal operando_A       : std_logic_vector(15 downto 0) := (others => '0');
signal operando_B       : std_logic_vector(15 downto 0) := (others => '0');
signal oper_ALU         : std_logic_vector(4 downto 0) := (others => '0');
signal p_ALU          : std_logic_vector(15 downto 0) := (others => '0');
signal sel_mux_q        : std_logic_vector(1 downto 0) := (others => '0');
signal q_ALU          : std_logic_vector(15 downto 0) := (others => '0');
signal cIN_ALU          : std_logic := '0';
signal out_ARI          : std_logic_vector(16 downto 0) := (others => '0');
signal out_LOG          : std_logic_vector(15 downto 0) := (others => '0');
signal out_SHIFT        : std_logic_vector(16 downto 0) := (others => '0');
signal sel_mux_LOG        : std_logic_vector(2 downto 0) := (others => '0');
signal sel_mux_ALU        : std_logic_vector(1 downto 0) := (others => '0');
signal aux_sel_mux_ALU_bit0   : std_logic := '0';
signal aux_FLAGMUX        : std_logic := '0';
signal aux_flagtest_rel     : std_logic := '0';
signal TRANS_OP         : std_logic_vector(1 downto 0) := (others => '0');
signal TRANS_FI_COND_IN     : std_logic_vector(3 downto 0) := (others => '0');
signal FLAGTEST_active_IN     : std_logic := '0';
signal aux_EXMEM_bit15      : std_logic := '0';
signal Sign_FLAG        : std_logic_vector(1 downto 0) := (others => '0');
signal aux_Sign_FLAG      : std_logic_vector(1 downto 0) := (others => '0');
signal out_ADD_MEM_aux      : std_logic_vector(11 downto 0) := (others => '0');
signal out_WE_MEM_aux     : std_logic := '0';


signal aux_reg_EXMEM_OUT_PCadd1   : std_logic_vector(11 downto 0) := (others => '0');
signal aux_reg_EXMEM_OUT_AddRWC   : std_logic_vector(2 downto 0) := (others => '0');
signal aux_reg_EXMEM_OUT_OutALU   : std_logic_vector(15 downto 0) := (others => '0');
signal aux_reg_EXMEM_OUT_OutMEM   : std_logic_vector(15 downto 0) := (others => '0');
signal aux_reg_EXMEM_OUT_MuxConst : std_logic_vector(15 downto 0) := (others => '0');
signal aux_reg_EXMEM_OUT_ovWE   : std_logic :='0';
signal aux_reg_EXMEM_OUT_SelMuxWB : std_logic_vector(1 downto 0) := (others => '0');

------------------------------------------------------------------------
--------------------  Constantes   -------------------------------------
------------------------------------------------------------------------
constant zero_12          : std_logic_vector(11 downto 0) := (others => '0');
constant zero_16          : std_logic_vector(15 downto 0) := (others => '0');
constant zeros_4          : std_logic_vector(3 downto 0) :=(others => '0');
constant menusum          : std_logic_vector(15 downto 0) :=(others => '1');
constant zeros_ALU        : std_logic_vector(15 downto 0) := (others => '0');

begin

operando_A        <= reg_IDOF_OUT_OperA;
operando_B        <= reg_IDOF_OUT_OperB;
oper_ALU        <= reg_IDOF_OUT_ALUOPER;
```

```
aux_EXMEM_bit15      <= reg_IDOF_OUT_bit15;

--
    ------------------------------------------------------------------------------------------------

---------------------------------- MEMORIA
    ----------------------------------------------------
--
    ------------------------------------------------------------------------------------------------


out_ADD_MEM_aux <= operando_A(11 downto 0); -- para enderecar leitura e escrita da RAM

out_ADD_MEM <= out_ADD_MEM_aux;

out_WE_MEM <=  reg_IDOF_OUT_WERAM;
out_MEM <= out_RAM; -- armazenar depois em RC o valor contido na posicao de memoria
    enderecada por A

in_RAM <= operando_B; -- armazenar na posicao de memoria enderecada por A o valor
    contido em B

--
    ------------------------------------------------------------------------------------------------

----------------------------------- ALU
    ----------------------------------------------------
--
    ------------------------------------------------------------------------------------------------


----------------------------------- aritmeticas
    ---------------------------------------------

p_ALU <= operando_A;

sel_mux_q <= oper_ALU(2) & oper_ALU(1);

q_ALU <= operando_B      when sel_mux_q = "00" else
     zeros_ALU       when sel_mux_q = "01" else
     not(operando_B)  when sel_mux_q = "10" else
     menusum;

cIN_ALU <= oper_ALU(0);

out_ARI <= ('0' & p_ALU) + ('0' & q_ALU) + cIN_ALU;

```

```
142 ------------------------------------- logicas
    -------------------------------------------
143
144 sel_mux_LOG <= (oper_ALU(3) xor oper_ALU(2)) & (oper_ALU(3) xor oper_ALU(1)) & (
    oper_ALU(3) xor oper_ALU(0));
145
146 out_LOG <=  zeros_ALU                 when sel_mux_LOG = "000" else
147       operando_A and operando_B         when sel_mux_LOG = "001" else
148       not(operando_A) and operando_B        when sel_mux_LOG = "010" else
149       operando_B                  when sel_mux_LOG = "011" else
150       operando_A and not(operando_B)        when sel_mux_LOG = "100" else
151       operando_A                  when sel_mux_LOG = "101" else
152       operando_A xor operando_B         when sel_mux_LOG = "110" else
153       operando_A or operando_B;
154
155 ------------------------------------- shifts
    -------------------------------------------
156
157 out_SHIFT <= (operando_A(15 downto 0) & '0') when oper_ALU(0) = '0' else  -- SLL
158       (operando_A(15) & operando_A(15) & operando_A(15 downto 1)); -- SRA
159
160 --------------------------------- resultado final da ALU
    ---------------------------------
161
162 sel_mux_ALU <= oper_ALU(4) & oper_ALU(3);
163
164 out_ALU <=  out_ARI(15 downto 0)      when sel_mux_ALU = "00"   else
165        out_SHIFT(15 downto 0)    when sel_mux_ALU = "01"   else
166         out_LOG                   when sel_mux_ALU = "10"    else
167         not(out_LOG);
168
169 --
    ------------------------------------------------------------------------------------------


170 ---------------------------------- FLAGS
    ------------------------------------------------
171 --
    ------------------------------------------------------------------------------------------


172
173 --------------------------------QUAIS FLAGS ATUALIZAM
    ??--------------------------------------
174
175 aux_Sign_FLAG(1) <=  not(oper_ALU(4));
176
177 aux_Sign_FLAG(0) <=  (oper_ALU(2) and (not(oper_ALU(1)) or oper_ALU(2))) or (oper_ALU
    (4) and (not(oper_ALU(2)) and ((not(oper_ALU(1)) and oper_ALU(0)) or oper_ALU(3)))
```

```
    ) or ((oper_ALU(4)) nor oper_ALU(3)) or (not(oper_ALU(0)) and oper_ALU(1));
178
179 Sign_FLAG <=  aux_Sign_FLAG  when (aux_EXMEM_bit15 and NOT(reg_IDOF_OUT_bit14) and NOT
    (reg_IDOF_OUT_ALUvsMEM)) = '1' else  "00";
180 --Actualizar FLAGS
181
182 ----------------------------FLAGS DA ARI-------------------------------
183
184
185 --OVERFLOW
186 aux_FLAGS_ARI(0) <=  (q_ALU(15) xnor p_ALU(15)) and (q_ALU(15) xor out_ALU(15));
187
188 --CARRY
189 aux_FLAGS_ARI(1) <=  out_ARI(16);
190
191 --NEGATIVE
192 aux_FLAGS_ARI(2) <=  out_ARI(15);
193
194 --ZERO
195 aux_FLAGS_ARI(3) <= not(out_ARI(15) or out_ARI(14) or out_ARI(13) or out_ARI(12)or
    out_ARI(11)
196          or out_ARI(10) or out_ARI(9) or out_ARI(8) or out_ARI(7) or out_ARI(6)
197          or out_ARI(5) or out_ARI(4) or out_ARI(3) or out_ARI(2) or out_ARI(1)
198          or out_ARI(0));
199
200 -------------------------FLAGS LOGICA-------------------------------
201 --NEGATIVE
202 aux_FLAGS_LOG(0) <= out_LOG(15);
203
204 --ZERO
205 aux_FLAGS_LOG(1) <= not(out_LOG(15) or out_LOG(14) or out_LOG(13) or out_LOG(12)or
    out_LOG(11)
206          or out_LOG(10) or out_LOG(9) or out_LOG(8) or out_LOG(7) or out_LOG(6)
207          or out_LOG(5) or out_LOG(4) or out_LOG(3) or out_LOG(2) or out_LOG(1)
208          or out_LOG(0));
209
210
211 ------------------------FLAGS SHIFT-------------------------------
212 --CARRY
213 aux_FLAGS_SHIFT(0) <= out_SHIFT(16);
214
215 --NEGATIVE
216 aux_FLAGS_SHIFT(1) <= out_SHIFT(15);
217
218 --ZERO
219 aux_FLAGS_SHIFT(2)  <= not(out_SHIFT(15) or out_SHIFT(14) or out_SHIFT(13) or
    out_SHIFT(12)or out_SHIFT(11)
```

```
220           or out_SHIFT(10) or out_SHIFT(9) or out_SHIFT(8) or out_SHIFT(7) or
    out_SHIFT(6)
221           or out_SHIFT(5) or out_SHIFT(4) or out_SHIFT(3) or out_SHIFT(2) or out_SHIFT
    (1)
222           or out_SHIFT(0));
223 ------------------------------------------------------------------
224
225 aux_FLAGS <= FLAGS_IN                    when Sign_FLAG ="00" else
226          aux_FLAGS_LOG & FLAGS_IN(1 downto 0)    when Sign_FLAG ="01" else
227          aux_FLAGS_SHIFT & FLAGS_IN(0)       when Sign_FLAG ="10" else
228          aux_FLAGS_ARI;
229
230
231
232
233 --
    ------------------------------------------------------------------------------------------
234 ---------------------------------- REGISTO FLAGS
    ------------------------------------------
235 --
    ------------------------------------------------------------------------------------------
236
237 process (clk, rst)
238   begin
239     if clk'event and clk = '1' then
240       if rst = '1' then
241         aux_MSR_FLAGS <= zeros_4;
242       else
243         aux_MSR_FLAGS <= aux_FLAGS;
244       end if;
245     end if;
246 end process;
247
248 Forw_FLAGSTEST_OUT <= aux_FLAGS;
249 FLAGS_OUT     <= aux_MSR_FLAGS;
250 FLAGSTEST_OUT <= aux_MSR_FLAGS;
251
252
253
254 ------------------------------------------------------------------------
255 ------------------------------ Exit ------------------------------------
256 ------------------------------------------------------------------------
257
258 --------------- registo de saida do terceiro andar: EX e MEM -------------
259 process (clk, rst)
```

```
  begin
    if clk'event and clk = '1' then
      if rst = '1' then
        aux_reg_EXMEM_OUT_ovWE     <= '0';
        aux_reg_EXMEM_OUT_SelMuxWB <= "00";
        aux_reg_EXMEM_OUT_PCadd1   <= zero_12;
        aux_reg_EXMEM_OUT_AddRWC   <= "000";
        aux_reg_EXMEM_OUT_OutALU   <= zero_16;
        aux_reg_EXMEM_OUT_OutMEM   <= zero_16;
        aux_reg_EXMEM_OUT_MuxConst <= zero_16;
      elsif en_EX = '1' then
        aux_reg_EXMEM_OUT_ovWE     <= reg_IDOF_OUT_ovWE;
        aux_reg_EXMEM_OUT_SelMuxWB <= reg_IDOF_OUT_SelMuxWB;
        aux_reg_EXMEM_OUT_PCadd1   <= reg_IDOF_OUT_PCadd1;
        aux_reg_EXMEM_OUT_AddRWC   <= reg_IDOF_OUT_AddRWC;
        aux_reg_EXMEM_OUT_OutALU   <= out_ALU;
        aux_reg_EXMEM_OUT_OutMEM   <= out_MEM;
        aux_reg_EXMEM_OUT_MuxConst <= reg_IDOF_OUT_MuxConst;
        -- reg_EXMEM_OUT <= aux_EXMEM_bit6 & aux_EXMEM_bit15 & out_MEM & ALU_vs_MEM &
    save_pc_add_1 & JUMP_MUXWB_OUT & aux_ADD_RWC &
        --                  out_ALU & out_mux_constantes & ALU_CONS_SEL;
      end if;
    end if;
end process;

  reg_EXMEM_OUT_ovWE      <= aux_reg_EXMEM_OUT_ovWE;
  reg_EXMEM_OUT_SelMuxWB    <= aux_reg_EXMEM_OUT_SelMuxWB;
  reg_EXMEM_OUT_PCadd1      <= aux_reg_EXMEM_OUT_PCadd1;
  reg_EXMEM_OUT_AddRWC      <= aux_reg_EXMEM_OUT_AddRWC;
  reg_EXMEM_OUT_OutALU      <= aux_reg_EXMEM_OUT_OutALU;
  reg_EXMEM_OUT_OutMEM      <= aux_reg_EXMEM_OUT_OutMEM;
  reg_EXMEM_OUT_MuxConst    <= aux_reg_EXMEM_OUT_MuxConst;
  ---------------
  ---Conflito----
  ---------------
  ADD_RWC_EXMEM <= reg_IDOF_OUT_AddRWC;
  ovWE_EXMEM    <= reg_IDOF_OUT_ovWE;


  Forw_EXMEN    <=  out_ALU  when reg_IDOF_OUT_SelMuxWB = "00" else
              out_MEM   when reg_IDOF_OUT_SelMuxWB = "01" else
              reg_IDOF_OUT_MuxConst when reg_IDOF_OUT_SelMuxWB = "10" else
              X"0"&reg_IDOF_OUT_PCadd1;


end Behavioral;
```

7.4 Código do andar de WB

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity WB is
  port(
    -- input
    clk, rst          : in std_logic;
    reg_EXMEM_OUT_PCadd1    : in std_logic_vector(11 downto 0);
    reg_EXMEM_OUT_AddRWC    : in std_logic_vector(2 downto 0);
    reg_EXMEM_OUT_OutALU    : in std_logic_vector(15 downto 0);
    reg_EXMEM_OUT_OutMEM    : in std_logic_vector(15 downto 0);
    reg_EXMEM_OUT_MuxConst    : in std_logic_vector(15 downto 0);
    reg_EXMEM_OUT_ovWE      : in std_logic;
    reg_EXMEM_OUT_SelMuxWB    : in std_logic_vector(1 downto 0);


    -- output
    ADD_RWC_WB          : out std_logic_vector(2 downto 0);
    ovWE_WB           : out std_logic;
    Forw_WB           : out std_logic_vector(15 downto 0);
    out_mux_WB          : out std_logic_vector(15 downto 0);
    out_saida         : out std_logic_vector(15 downto 0);
    en_regs           : out std_logic_vector(7 downto 0)
  );
end WB;

architecture Behavioral of WB is

---------------------------------------------------------------------------
--------------------------- Aux Signals -----------------------------------
---------------------------------------------------------------------------
signal sel_mux_WB       : std_logic_vector(1 downto 0) := (others => '0');
signal aux_out_alu        : std_logic_vector(15 downto 0):= (others => '0');
signal aux_out_const      : std_logic_vector(15 downto 0):= (others => '0');
signal aux_mux_WB     : std_logic_vector(15 downto 0):= (others => '0');
signal aux_out_pcadd1     : std_logic_vector(15 downto 0):= (others => '0');
```

```
signal aux_en_WC        : std_logic_vector(2 downto 0):= (others => '0');


constant zeros        : std_logic_vector(15 downto 0) := (others => '0');
begin


sel_mux_WB <= reg_EXMEM_OUT_SelMuxWB;

aux_out_alu    <= reg_EXMEM_OUT_OutALU;
aux_out_const    <= reg_EXMEM_OUT_MuxConst;
aux_out_pcadd1   <= X"0" & reg_EXMEM_OUT_PCadd1;

aux_mux_WB <=   aux_out_alu               when sel_mux_WB = "00" else     -- escrever
    a saida da ALU    (out_ALU)
             reg_EXMEM_OUT_OutMEM         when sel_mux_WB = "01" else     -- escrever
    saida MEM       (out_MEM)
             aux_out_const              when sel_mux_WB = "10" else   -- fazer load de
     uma constante    (out_mux_constantes)
            aux_out_pcadd1;                                    -- guardar em R7 o valor
     de PC+1  (save_pc_add_1)

out_mux_WB <= aux_mux_WB;
aux_en_WC <= "111" when (sel_mux_WB(1) and sel_mux_WB(0)) = '1' else
    reg_EXMEM_OUT_AddRWC;



with aux_en_WC select
    en_regs <=  "0000000" & reg_EXMEM_OUT_ovWE      when "000",
          "000000" & reg_EXMEM_OUT_ovWE & '0'   when "001",
          "00000" & reg_EXMEM_OUT_ovWE & "00"   when "010",
          "0000" & reg_EXMEM_OUT_ovWE & "000"   when "011",
          "000" & reg_EXMEM_OUT_ovWE & "0000"   when "100",
          "00" & reg_EXMEM_OUT_ovWE & "00000"   when "101",
          '0' & reg_EXMEM_OUT_ovWE & "000000"   when "110",
          reg_EXMEM_OUT_ovWE & "0000000"       when "111",
          "00000000"              when others;


-----------------------------------
------------EXIT-------------------

ADD_RWC_WB  <= reg_EXMEM_OUT_AddRWC;
ovWE_WB   <= reg_EXMEM_OUT_ovWE;

Forw_WB <= aux_mux_WB;
```

```
88
89 end Behavioral;
```

7.5 Código da memória ROM

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.STD_LOGIC_UNSIGNED.ALL;
4  use STD.TEXTIO.all;
5  use STD.TEXTIO;
6  use IEEE.STD_LOGIC_TEXTIO.all;
7
8  entity memoria_ROM is
9      Generic(
10       ADDR_SIZE : positive:= 12
11     );
12     Port(
13       Addr_ROM  : in  STD_LOGIC_VECTOR(ADDR_SIZE-1 downto 0);
14       DO_ROM  : out   STD_LOGIC_VECTOR(15 downto 0)
15     );
16 end memoria_ROM;
17
18 architecture Behavioral of memoria_ROM is
19
20 type MEM_TYPE is array(0 to (2**ADDR_SIZE)-1) of STD_LOGIC_VECTOR(15 downto 0);
21
22  impure function InitRamFromFile (RamFileName : in string) return MEM_TYPE is
23     file INFILE : TEXT  is in "rom_inst.txt";
24     variable DATA_TEMP : STD_LOGIC_VECTOR(15 downto 0);
25     variable IN_LINE: LINE;
26     variable ROM : MEM_TYPE;
27     variable index :integer;
28
29     begin
30         index := 0;
31         while NOT(endfile(INFILE)) loop
32           readline(INFILE,IN_LINE);
33           hread(IN_LINE, DATA_TEMP);
34           ROM(index) := DATA_TEMP;
35           index := index + 1;
36         end loop;
37         for index in index to 4095 loop
38           ROM(index) := X"0000";
39         end loop;
40   return ROM;
41    end function;
42
43 signal ROM : MEM_TYPE := InitRamFromFile("rom_inst.txt");
```

```
44
45
46 begin
47
48 DO_ROM <= ROM(conv_integer(Addr_ROM)); -- leitura assincrona
49
50 end Behavioral;
```

7.6 Código da memória RAM

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.STD_LOGIC_UNSIGNED.ALL;
4  use STD.TEXTIO.all;
5  use STD.TEXTIO;
6  use IEEE.STD_LOGIC_TEXTIO.all;
7
8
9  entity memoria_RAM is
10     Generic(
11       ADDR_SIZE : positive:= 12
12     );
13     Port(
14       CLK_A    : in  STD_LOGIC;
15       WE_A     : in  STD_LOGIC;
16       Addr_A   : in  STD_LOGIC_VECTOR(ADDR_SIZE-1 downto 0);
17       DI_A   : in  STD_LOGIC_VECTOR(15 downto 0);
18       DO_A   : out   STD_LOGIC_VECTOR(15 downto 0)
19     );
20 end memoria_RAM;
21
22 architecture Behavioral of memoria_RAM is
23
24 type MEM_TYPE is array(0 to (2**ADDR_SIZE)-1) of STD_LOGIC_VECTOR(15 downto 0);
25
26 impure function InitRamFromFile (RamFileName : in string) return MEM_TYPE is
27     file INFILE : TEXT  is in "ram_inst.txt";
28     variable DATA_TEMP : STD_LOGIC_VECTOR(15 downto 0);
29     variable IN_LINE: LINE;
30     variable RAM : MEM_TYPE;
31     variable index :integer;
32     variable i :integer;
33
34     begin
35         index := 0;
36         i:=0;
37         readline(INFILE,IN_LINE);
38         hread(IN_LINE, DATA_TEMP);
```

```
        index := CONV_INTEGER(DATA_TEMP);

        for i in i to index loop
          RAM(i) := X"0000";
        end loop;
        while NOT(endfile(INFILE)) loop
          readline(INFILE,IN_LINE);
          hread(IN_LINE, DATA_TEMP);
          RAM(index) := DATA_TEMP;
          index := index + 1;
        end loop;
        for index in index to 4095 loop
          RAM(index) := X"0000";
        end loop;
  return RAM;
   end function;

shared variable RAM : MEM_TYPE := InitRamFromFile("ram_inst.txt");

begin
process (CLK_A)
  begin
    if rising_edge(CLK_A)then
      if WE_A='1'  then
        RAM(conv_integer(Addr_A)) := DI_A;
      end if;
  end if;
end process;

DO_A <= RAM(conv_integer(Addr_A)); -- leitura assincrona

end Behavioral;
```

7.7 Código da BTB

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use STD.TEXTIO.all;
use STD.TEXTIO;
use IEEE.STD_LOGIC_TEXTIO.all;


entity BTB_bram is
    Generic(
      ADDR_SIZE : positive:= 9
    );
    Port(
```

```
      CLK_A    : in  STD_LOGIC;
      WE_A     : in  STD_LOGIC;
      Addr_A   : in  STD_LOGIC_VECTOR(ADDR_SIZE-1 downto 0);
      Addr_B   : in  STD_LOGIC_VECTOR(ADDR_SIZE-1 downto 0);
      DI_A     : in  STD_LOGIC_VECTOR(16 downto 0);
      DO_B     : out STD_LOGIC_VECTOR(16 downto 0)
    );
end BTB_bram;

architecture Behavioral of BTB_bram is

type MEM_TYPE is array(0 to (2**ADDR_SIZE)-1) of STD_LOGIC_VECTOR(16 downto 0);

constant InitValue : MEM_TYPE := ( others => "00000000000000000");

shared variable myRAM : MEM_TYPE := InitValue;

begin
process (CLK_A)
  begin
    if rising_edge(CLK_A)then
      if WE_A='1'  then
        myRAM(conv_integer(Addr_A)) := DI_A;
      end if;
  end if;
end process;

DO_B <= myRAM(conv_integer(Addr_B)); -- leitura assincrona

end Behavioral;
```

7.8 Código da máquina de estados

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_SIGNED.ALL;


entity Controlo is
  port (
      clk, rst  : in std_logic;
      en_if     : out std_logic;
      en_idOF   : out std_logic;
      en_EX   : out std_logic
      );
end Controlo;
```

```
16
17
18 architecture Behavioral of Controlo is
19     type fsm_states is (s_inicial,s_inicial2,s_inicial3,s_cont);
20     signal currstate, nextstate: fsm_states;
21 begin
22
23 state_reg: process (clk,rst)
24     begin
25      if rst = '1' then
26       currstate <= s_inicial ;
27      elsif clk'event and clk = '1' then
28       currstate <= nextstate ;
29      end if ;
30 end process;
31
32 state_comb: process (currstate )
33   begin  --  process
34    nextstate <= currstate;
35
36
37     case currstate is
38     when s_inicial  =>
39       nextstate <= s_inicial2;
40       en_if     <= '1';
41       en_idOF   <= '1';
42       en_EX     <= '1';
43     when s_inicial2 =>
44       nextstate <= s_inicial3;
45       en_if     <= '1';
46       en_idOF   <= '1';
47       en_EX     <= '1';
48     when s_inicial3 =>
49       nextstate <= s_cont;
50       en_if     <= '1';
51       en_idOF   <= '1';
52       en_EX     <= '1';
53
54       when s_cont =>
55         nextstate <= s_cont ;
56           en_if     <= '1';
57         en_idOF   <= '1';
58         en_EX     <= '1';
59
60     end case;
61   end process;
62
```

```
63
64 end Behavioral;
```

Todo list