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)*.

2 Conflitos associados a uma arquitectura *pipeline*

2.1 Conflitos estruturais

Um conflito estrutural é quando a estrutura de um processador não tem os recursos suficientes para executar uma sequência de instruções em *pipelining*. No entanto, o processador projectado não irá ter conflitos estruturais, isto porque terá recursos suficientes para executar todos os andares em *pipelinig*.

2.2 Conflitos de dados

Um conflito de dados ocorre quando o *pipelinig* muda a ordem de acesso a processos de escrita/leitura de operandos de modo que mude a ordem pretendida ou não permita obter o valor desejado

this sentence is weird

Quando se fala em conflitos de dados fala-se de três tipos distintos:

- *Read After Write* (RAW): ocorre quando uma instrução precisa de ler um valor que ainda não foi escrito na memória, pois pertence a uma instrução anterior que ainda não escreveu no seu registo de destino;
- *Write After Read* (WAR): ocorre quando uma instrução necessita de escrever num registo, numa altura em que a instrução anterior ainda não leu o valor desse registo e necessita;
- *Write After Write* (WAW): ocorre quando duas operações necessitam de escrever no mesmo registo ao mesmo tempo ou numa ordem incorrecta;

No processador μRisc a projectar apenas ocorrem conflitos do tipo RAW. De facto, os conflitos do tipo WAR e do tipo WAW não ocorrem no processador a projectar, pois não existe qualquer tipo de *bypassing* entre os seus andares, mantendo-se sempre a ordem das instruções intacta.

2.3 Conflitos de controlo

Quando uma instrução de controlo condicional é executada, esta tem duas possibilidades - ou altera o valor de PC (*program counter*) para a próxima instrução (salto *not taken*) ou para onde a instrução pretendida (salto *taken*).

3 Métodos de resolução de conflitos

3.1 Conflitos de dados

- 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 e se for feita por *hardware* é um escalonamento dinâmico;

Analisando as soluções apresentadas verificou-se que o escalonamento estático e dinâmico não é a solução desejada devido a complexidade para um processador de 4 andares comparativamente às soluções expostas 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 (IPC), optou-se pela segunda solução que permite aumentar o valor de IPC com a contrapartida de ter uma maior complexidade de implementação.

3.2 Conflitos de controlo

- Solução 1: *Branch Target Buffer* (BTB) que é uma tabela que contém informação sobre os saltos já executados com o objectivo de tentar prever se uma nova instrução de salto é *taken* ou *not-taken*. A previsão é realizada por uma *Branch Predict Buffer* (BPB), que é composta por 1 ou 2 *bits*, e, caso a predição esteja correcta, diminui o número de *stalls* necessários e, por sua vez, o número de ciclos de execução do programa;
- Solução 2: *forwarding* de *flags*, ou seja, lógica adicional que permite verificar se a previsão feita anteriormente pela BTB está correcta ou não, consoante a condição de salto;

Em análise às duas soluções apresentadas foi decidido que ambas seriam usadas no processador. Em relação à BTB, decidiu-se usar uma BPB de 1 *bit*, pois não se achou necessário prever saltos com a profundidade de *strong taken* e *strong not-taken*.

De referir que, ao acrescentar *forwarding* de *flags*, o processador fica mais complexo mas, para verificar se a previsão foi efectuada correctamente, não é necessário aguardar dois ciclos mas sim apenas um, o que compensa para o valor de IPC.

4 Estrutura do processador

Como foi referido anteriormente, os conflitos de dados são resolvidos por *forwarding* de dados e os conflitos de controlo são resolvidos por BTB e *forwarding* de flags. De seguida, estão representados os circuitos e sinais que representam as unidades de resolução de conflitos.

4.1 Circuitos e sinais na resolução de conflitos de dados

A solução desejada é a aplicação de *forwarding* de dados e este divide-se em duas partes: detecção de conflitos de dados e encaminhamento do resultado pretendido no andar que gera o conflito.

Nas figuras seguintes apresenta-se a lógica responsável para a detecção dos conflitos de dados.

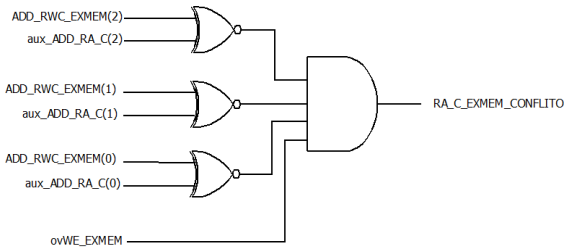

Figura 1: Detecção dos conflitos de dados para o operando A, no andar de EX/MEM.

Este circuito anterior representa a detecção de conflito de dados referente ao andar EX/MEM e consiste em comparar o endereço de escrita com o endereço do registo pretendido. Se estes forem iguais, o sinal `RA_C_EXMEM_CONFLITO` tem valor lógico a 1 se o registo pretendido for o operando A ou o sinal `RB_EXMEM_CONFLITO`, que segue uma lógica equivalente à anterior, tem valor lógico a 1 se o registo pretendido for o operando B.

O circuito seguinte verifica se a instrução não é dependente de registos, para o registo A. É o caso de uma instrução de *load* de constantes em que a constante a ser guardada pode, erradamente, endereçar um registo em que o valor pode estar no *pipeline*.

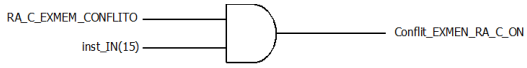

Figura 2: Detecção dos conflitos de dados para o operando B, no andar de EX/MEM.

O circuito seguinte tem o mesmo objectivo que o circuito anterior mas para o registo B, pois existem instruções de salto que utilizam o registo B para efectuar os saltos relativos, como é o caso das instruções *jump-and-link* e *jump register*. Obtém-se assim dois sinais - um referente ao registo B, `Conflit_EXMEM_RB_ON` e outro referente ao registo A, `Conflit_EXMEM_RA_C_ON`.

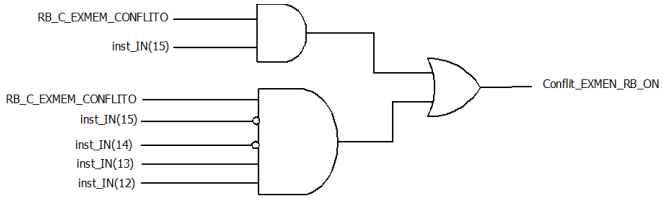

Figura 3: Circuito que permite verificar se uma instrução está associada à leitura de registos.

O circuito seguinte representa a detecção de conflito de dados referente ao andar WB, *forwarding* é necessário pois o banco de registo é síncrono, como na situação anterior consiste em verificar o endereço de escrita com o endereço do registo pretendido, se estes forem iguais o sinal `RA_C_WB_CONFLITO` tem sinal lógico a 1 se o registo pretendido for o operando A ou o sinal `RB_WB_CONFLITO` tem sinal lógico a 1 se o registo pretendido for o operando B.

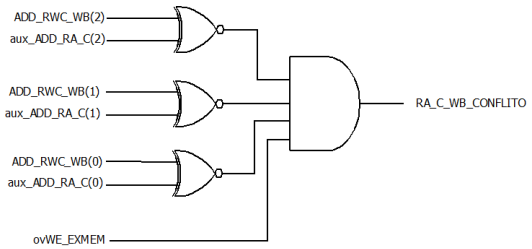

Figura 4: Circuito que permite verificar se uma instrução está associada à leitura de registos.

Relativamente à detecção no andar de WB também é aplicado o circuito TAL para a detecção dos conflitos com registo A e o circuito TAL para os conflitos com o registo B. Obtém-se assim dois sinais, um referente ao registo B, `Conflit_WB_RB_ON` e outro referente ao registo A, `Conflit_WB_RA_C_ON`.

Gerando assim dois sinais de controlo que seleccionam os valores dos operandos A e B, o sinal `mux_RA` sinal de 2 *bits* que consiste na concatenação do sinal`Conflit_WB_RA_C_ON` com `Conflit_EXMEM_RA_C_ON`, e o sinal `mux_RB` sinal de 2 *bits* que consiste na concatenação do sinal`Conflit_WB_RB_ON` com `Conflit_EXMEM_RB_ON`.

De seguida esta representado o esquema geral de *forwarding* nos andares de *pipelining*.

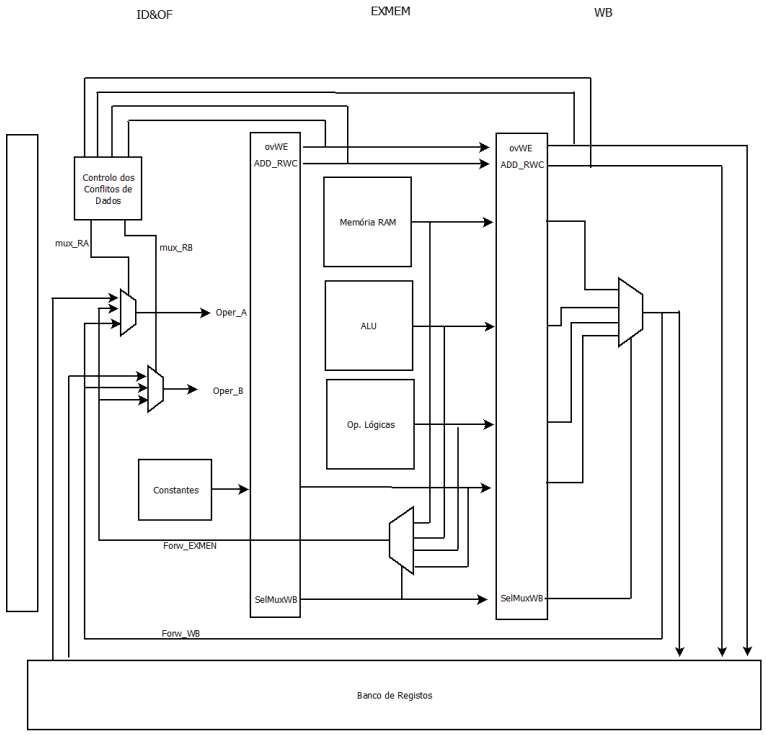

Figura 5: Esquema geral de resolução de conflitos de dados.

É de referir que foi acrescentado um MUX no andar EX/MEM de forma a obter o valor correcto do registo de escrita.

4.2 Circuitos e sinais na resolução de conflitos de controlo

5 Testes de *performance*

Após o projecto do processador estar concluído procede-se à fase de testes. As métricas utilizadas foram: a frequência obtida com base no caminho crítico, o número de ciclos de execução até que o programa corra por completo, o tempo de execucão e o número de predicções falhadas por parte da BTB. Na tabela seguinte encontram-se os resultados obtidos para os três testes fornecidos.

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

De referir que o caminho crítico do processador passa pelo circuito responsável pelo *forwarding* de *flags*, como se pode ver na próxima imagem.

```
Timing constraint: Default period analysis for Clock 'clk_in'
  Clock period: 5.459ns (frequency: 183.169MHz)
  Total number of paths / destination ports: 211316 / 470
-------------------------------------------------------------------------
Delay:               5.459ns (Levels of Logic = 23)
  Source:            inst_IDeOF/aux_reg_IDOF_OUT_OperB_0 (FF)
  Destination:       inst_BTB_BRAM/Mram_myRAM (RAM)
  Source Clock:      clk_in rising
  Destination Clock: clk_in rising

  Data Path: inst_IDeOF/aux_reg_IDOF_OUT_OperB_0 to inst_BTB_BRAM/Mram_myRAM
                                Gate     Net
    Cell:in->out      fanout   Delay   Delay  Logical Name (Net Name)
    ----------------------------------------  ------------
     FDRE:C->Q             3   0.254   0.560  inst_IDeOF/aux_reg_IDOF_OUT_OperB_0 (inst_IDeOF/aux_reg_IDOF_OUT_OperB_0)
     LUT4:I0->O            1   0.049   0.000  inst_EXeMEM/Madd_out_ARI_Madd_lut<0> (inst_EXeMEM/Madd_out_ARI_Madd_lut<0>)
     MUXCY:S->O            1   0.257   0.000  inst_EXeMEM/Madd_out_ARI_Madd_cy<0> (inst_EXeMEM/Madd_out_ARI_Madd_cy<0>)
     MUXCY:CI->O           1   0.014   0.000  inst_EXeMEM/Madd_out_ARI_Madd_cy<1> (inst_EXeMEM/Madd_out_ARI_Madd_cy<1>)
     MUXCY:CI->O           1   0.014   0.000  inst_EXeMEM/Madd_out_ARI_Madd_cy<2> (inst_EXeMEM/Madd_out_ARI_Madd_cy<2>)
     MUXCY:CI->O           1   0.014   0.000  inst_EXeMEM/Madd_out_ARI_Madd_cy<3> (inst_EXeMEM/Madd_out_ARI_Madd_cy<3>)
     MUXCY:CI->O           1   0.014   0.000  inst_EXeMEM/Madd_out_ARI_Madd_cy<4> (inst_EXeMEM/Madd_out_ARI_Madd_cy<4>)
     MUXCY:CI->O           1   0.014   0.000  inst_EXeMEM/Madd_out_ARI_Madd_cy<5> (inst_EXeMEM/Madd_out_ARI_Madd_cy<5>)
     MUXCY:CI->O           1   0.014   0.000  inst_EXeMEM/Madd_out_ARI_Madd_cy<6> (inst_EXeMEM/Madd_out_ARI_Madd_cy<6>)
     MUXCY:CI->O           1   0.014   0.000  inst_EXeMEM/Madd_out_ARI_Madd_cy<7> (inst_EXeMEM/Madd_out_ARI_Madd_cy<7>)
     MUXCY:CI->O           1   0.014   0.000  inst_EXeMEM/Madd_out_ARI_Madd_cy<8> (inst_EXeMEM/Madd_out_ARI_Madd_cy<8>)
     MUXCY:CI->O           1   0.014   0.000  inst_EXeMEM/Madd_out_ARI_Madd_cy<9> (inst_EXeMEM/Madd_out_ARI_Madd_cy<9>)
     MUXCY:CI->O           1   0.014   0.000  inst_EXeMEM/Madd_out_ARI_Madd_cy<10> (inst_EXeMEM/Madd_out_ARI_Madd_cy<10>)
     MUXCY:CI->O           1   0.014   0.000  inst_EXeMEM/Madd_out_ARI_Madd_cy<11> (inst_EXeMEM/Madd_out_ARI_Madd_cy<11>)
     MUXCY:CI->O           1   0.014   0.000  inst_EXeMEM/Madd_out_ARI_Madd_cy<12> (inst_EXeMEM/Madd_out_ARI_Madd_cy<12>)
     MUXCY:CI->O           1   0.014   0.000  inst_EXeMEM/Madd_out_ARI_Madd_cy<13> (inst_EXeMEM/Madd_out_ARI_Madd_cy<13>)
     MUXCY:CI->O           1   0.014   0.000  inst_EXeMEM/Madd_out_ARI_Madd_cy<14> (inst_EXeMEM/Madd_out_ARI_Madd_cy<14>)
     XORCY:CI->O           4   0.282   0.368  inst_EXeMEM/Madd_out_ARI_Madd_xor<15> (inst_EXeMEM/out_ARI<15>)
     LUT6:I5->O            5   0.049   0.438  inst_EXeMEM/Mmux_aux_FLAGS31 (Forw_FLAGS_IN<2>)
     LUT6:I4->O            3   0.049   0.363  inst_IDeOF/aux_FLAGMUX_C_For1_SW1 (N123)
     LUT5:I4->O            1   0.049   0.640  inst_IDeOF/aux_FLAGTEST_MUXPC1_SW1 (N225)
     LUT6:I1->O           21   0.049   0.462  inst_IDeOF/aux_FLAGTEST_MUXPC1 (act_BTB)
     LUT6:I5->O            1   0.049   0.548  inst_InF/MUX_NEXTPC<1>1_SW1 (N145)
     LUT6:I2->O            1   0.049   0.339  inst_InF/aux_saida_mux<0>1 (inst_InF/aux_saida_mux<0>_0)
     RAMB18E1:ADDRBWRADDR4         0.416          inst_BTB_BRAM/Mram_myRAM
    ----------------------------------------
    Total                      5.459ns (1.741ns logic, 3.718ns route)
                                       (31.9% logic, 68.1% route)
```

Figura 6: Caminho crítico do processador com frequência de 183,169 MHz.

No entanto, já após a demonstração do processador na aula laboratorial e depois de uma análise mais cuidada do caminho crítico concluiu-se que ainda era possível efecutar uma melhoria na *performance* do μRisc. De facto, no caminho crítico existe um somador no andar de ID que pode passar para o andar anterior, ou seja, o andar de IF, o que permite aumentar a frequência do processador.

Desta maneira, o novo caminho crítico permite ter uma frequência de 211,338 MHz. Na figura seguinte apresenta-se o novo caminho crítico como gerado pela ferramenta.

```
================================================================================
Timing constraint: Default period analysis for Clock 'clk_in'
  Clock period: 4.732ns (frequency: 211.338MHz)
  Total number of paths / destination ports: 160209 / 677
--------------------------------------------------------------------------------
Delay:               4.732ns (Levels of Logic = 7)
  Source:            inst_IDeOF/aux_reg_IDOF_OUT_ALUOPER_0_1 (FF)
  Destination:       inst_BTB_BRAM/Mram_myRAM (RAM)
  Source Clock:      clk_in rising
  Destination Clock: clk_in rising

  Data Path: inst_IDeOF/aux_reg_IDOF_OUT_ALUOPER_0_1 to inst_BTB_BRAM/Mram_myRAM
                                Gate     Net
    Cell:in->out      fanout   Delay   Delay  Logical Name (Net Name)
    ----------------------------------------  ------------
     FDRE:C->Q             3   0.254   0.663  inst_IDeOF/aux_reg_IDOF_OUT_ALUOPER_0_1 (inst_IDeOF/aux_reg_IDOF_OUT_ALUOPER_0_1)
     LUT6:I0->O            2   0.049   0.656  inst_EXeMEM/Mmux_out_LOG81 (inst_EXeMEM/out_LOG<1>)
     LUT6:I0->O            1   0.049   0.351  inst_EXeMEM/Mmux_aux_FLAGS45 (inst_EXeMEM/Mmux_aux_FLAGS44)
     LUT5:I4->O            1   0.049   0.351  inst_EXeMEM/Mmux_aux_FLAGS47_SW1 (N355)
     LUT6:I5->O            1   0.049   0.351  inst_EXeMEM/Mmux_aux_FLAGS47 (inst_EXeMEM/Mmux_aux_FLAGS46)
     LUT6:I5->O            6   0.049   0.545  inst_EXeMEM/Mmux_aux_FLAGS417 (Forw_FLAGS_IN<3>)
     LUT6:I3->O           21   0.049   0.462  inst_InF/MUX_NEXTPC<0>1 (inst_InF/MUX_NEXTPC<0>)
     LUT6:I5->O            1   0.049   0.339  inst_InF/aux_saida_mux<0>_0_01 (inst_InF/aux_saida_mux<0>_0_0_0)
     RAMB18E1:ADDRBWRADDR4     0.416          inst_BTB_BRAM/Mram_myRAM
    ----------------------------------------
    Total                      4.732ns (1.013ns logic, 3.719ns route)
                                       (21.4% logic, 78.6% route)
```

Figura 7: Caminho crítico do processador com frequência de 211,338 MHz.

Para se perceber melhor a influência que os métodos utilizados para resolver os conflitos têm no desempenho do processador foram realizados vários testes com diversas topologias do mesmo processador. Assim, fez-se testes para o processador falado anteriormente (processador #1) e outros três processadores diferentes, cada um com 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

O processador #1 tem o caminho crítico da Figura 7, assim como o processador #2. Os processadores #3 e #4 tem o caminho crítico da figura apresentada de seguida.

```
================================================================================
Timing constraint: Default period analysis for Clock 'clk_in'
  Clock period: 3.501ns (frequency: 285.608MHz)
  Total number of paths / destination ports: 49076 / 540
--------------------------------------------------------------------------------
Delay:               3.501ns (Levels of Logic = 17)
  Source:            inst_InF/aux_pc_OUT_0 (FF)
  Destination:       inst_InF/aux_reg_pc_0 (FF)
  Source Clock:      clk_in rising
  Destination Clock: clk_in rising

  Data Path: inst_InF/aux_pc_OUT_0 to inst_InF/aux_reg_pc_0
                                Gate     Net
    Cell:in->out      fanout   Delay   Delay  Logical Name (Net Name)
    ----------------------------------------  ------------
     FDRE:C->Q             2   0.254   0.356  inst_InF/aux_pc_OUT_0 (inst_InF/aux_pc_OUT_0)
     LUT2:I1->O            1   0.049   0.000  inst_IDeOF/Mmux_Jump_BTB_rs_lut<0> (inst_IDeOF/Mmux_Jump_BTB_rs_lut<0>)
     MUXCY:S->O            1   0.257   0.000  inst_IDeOF/Mmux_Jump_BTB_rs_cy<0> (inst_IDeOF/Mmux_Jump_BTB_rs_cy<0>)
     MUXCY:CI->O           1   0.013   0.000  inst_IDeOF/Mmux_Jump_BTB_rs_cy<1> (inst_IDeOF/Mmux_Jump_BTB_rs_cy<1>)
     MUXCY:CI->O           1   0.013   0.000  inst_IDeOF/Mmux_Jump_BTB_rs_cy<2> (inst_IDeOF/Mmux_Jump_BTB_rs_cy<2>)
     MUXCY:CI->O           1   0.013   0.000  inst_IDeOF/Mmux_Jump_BTB_rs_cy<3> (inst_IDeOF/Mmux_Jump_BTB_rs_cy<3>)
     MUXCY:CI->O           1   0.013   0.000  inst_IDeOF/Mmux_Jump_BTB_rs_cy<4> (inst_IDeOF/Mmux_Jump_BTB_rs_cy<4>)
     MUXCY:CI->O           1   0.013   0.000  inst_IDeOF/Mmux_Jump_BTB_rs_cy<5> (inst_IDeOF/Mmux_Jump_BTB_rs_cy<5>)
     MUXCY:CI->O           1   0.013   0.000  inst_IDeOF/Mmux_Jump_BTB_rs_cy<6> (inst_IDeOF/Mmux_Jump_BTB_rs_cy<6>)
     MUXCY:CI->O           1   0.013   0.000  inst_IDeOF/Mmux_Jump_BTB_rs_cy<7> (inst_IDeOF/Mmux_Jump_BTB_rs_cy<7>)
     MUXCY:CI->O           1   0.013   0.000  inst_IDeOF/Mmux_Jump_BTB_rs_cy<8> (inst_IDeOF/Mmux_Jump_BTB_rs_cy<8>)
     MUXCY:CI->O           1   0.013   0.000  inst_IDeOF/Mmux_Jump_BTB_rs_cy<9> (inst_IDeOF/Mmux_Jump_BTB_rs_cy<9>)
     MUXCY:CI->O           0   0.013   0.000  inst_IDeOF/Mmux_Jump_BTB_rs_cy<10> (inst_IDeOF/Mmux_Jump_BTB_rs_cy<10>)
     XORCY:CI->O          20   0.282   0.457  inst_IDeOF/Mmux_Jump_BTB_rs_xor<11> (Jump_BTB<11>)
     LUT6:I5->O            1   0.049   0.517  inst_InF/Jump_BTB[11]_aux_pc_add_1[11]_equal_17_o125_SW8 (N72)
     LUT6:I3->O           12   0.049   0.476  inst_IDeOF/aux_FLAGMUX_C_For2_SW4 (N56)
     LUT5:I3->O            1   0.049   0.517  inst_InF/Mmux_aux_saida_mux121_SW0 (N142)
     LUT6:I3->O            1   0.049   0.000  inst_InF/Mmux_aux_saida_mux121 (inst_InF/aux_saida_mux<9>)
     FDRE:D                    0.004          inst_InF/aux_reg_pc_9
    ----------------------------------------
    Total                      3.501ns (1.177ns logic, 2.324ns route)
                                       (33.6% logic, 66.4% route)
```

Figura 8: Caminho crítico do processador com frequência de 285,608 MHz.

Nas tabelas seguintes apresentam-se as métricas obtidas para os vários processadores para os três testes fornecidos.

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.

5.1 Efeitos de *forwarding* de dados

Ao observar as características dos quatro processadores, percebe-se que é comparando os resultados entre os processadores #1 e #2 e também entre os processadores #3 e #4 que se consegue avaliar os efeitos de usar *forwarding* de dados. Os cálculos do *speed-up* são feitos através da equação 5.1.

$$\textit{speed-up} = \frac{N_{\text{ciclos}_{\text{processador\#2}}}}{N_{\text{ciclos}_{\text{processador\#1}}}} = \frac{N_{\text{ciclos}_{\text{processador\#4}}}}{N_{\text{ciclos}_{\text{processador\#3}}}} \tag{5.1}$$

Em relação ao primeiro teste (Tabela 3), foi possível estabelecer um *speed-up* de exactamente 2. Em relação ao segundo teste (Tabela 4) os *speed-up*'s alcançados foram de 1,291 e de 1,228, enquanto que no terceiro teste (Tabela 4) foram alcançados *speed-up*'s de 1,495 e de 1,467. Como nos três testes, tal como esperado, utilizar o método de *forwarding de dados* influenciou a frequência dos processadores, podemos admitir que este método obtém resultados muito positivos.

5.2 Efeitos de *branch prediction*

Ao observar as características dos quatro processadores, percebemos que é comparando os resultados entre os processadores #1 e #3 e também entre os processadores #2 e #4 que conseguimos avaliar os efeitos de usar uma BTB.

No primeiro teste (Tabela 3) foi possível observar que nem sempre é positivo ter uma BTB, pois esta aumenta o tempo do caminho crítico, diminuindo assim a frequência do processador, e caso o teste não tenha saltos, o que se observa é um *speed-up* menor que 1.

Em relação ao segundo teste (Tabela 4), foi possível verificar que um processador com uma BTB, embora mais lento, apenas falhou apenas 23,3% das predições, e que no terceiro teste falhou 65,2% das predições. No caso destes dois testes, este aumento de precisão não correspondeu a um tempo

de execução menor, no entanto, caso o teste fosse mais extenso seria possível observar um tempo de execução menor para um processador com *branch prediction*.

6 Conclusões

7 Anexos

Em anexo ao relatório encontra-se o código VHDL que implementa o processador em regime de *pipeline*.

Todo list