

INSTITUTO SUPERIOR TÉCNICO

MESTRADO INTEGRADO EM ENGENHARIA ELECTROTÉCNICA E DE  
COMPUTADORES

ARQUITECTURAS AVANÇADAS DE COMPUTADORES

**Descrição do processador  $\mu$ Risc 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

<b>1</b>	<b>Introdução</b>	<b>1</b>
<b>2</b>	<b>Conflitos associados a uma arquitectura <i>pipeline</i></b>	<b>1</b>
2.1	Conflitos estruturais . . . . .	1
2.2	Conflitos de dados . . . . .	1
2.3	Conflitos de controlo . . . . .	1
<b>3</b>	<b>Métodos de resolução de conflitos</b>	<b>2</b>
3.1	Conflitos de dados . . . . .	2
3.2	Conflitos de controlo . . . . .	2
<b>4</b>	<b>Estrutura do processador</b>	<b>2</b>
4.1	Circuitos e sinais na resolução de conflitos de dados . . . . .	3
4.2	Circuitos e sinais na resolução de conflitos de controlo . . . . .	3
<b>5</b>	<b>Testes de <i>performance</i></b>	<b>3</b>
5.1	Efeitos de <i>forwarding</i> de dados . . . . .	5
5.2	Efeitos de <i>branch prediction</i> . . . . .	5
<b>6</b>	<b>Conclusões</b>	<b>5</b>
<b>7</b>	<b>Anexos</b>	<b>6</b>

# 1 Introdução

Com este trabalho laboratorial pretende-se projectar um processador  $\mu$ Risc 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 *pipelining*.

### 2.2 Conflitos de dados

Um conflito de dados ocorre quando o *pipelining* 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

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  $\mu$ Risc 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*).

this sentence is weird

falar mais aqui

## 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 aplicação de *forwarding* de dados. O *forwarding* de dados 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,

Este circuito representa a detecção de conflito de dados referente ao andar EX/MEM, consiste em verificar o endereço de escrita com o endereço do registo pretendido, se estes forem iguais o sinal RA\_C\_EXMEM\_CONFLITO tem sinal lógico a 1 se o registo pretendido for o operando A ou o sinal RB\_EXMEM\_CONFLITO tem sinal lógico a 1 se o registo pretendido for o operando B.

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

Este circuito 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.

## 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 execução 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.

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  $\mu$ Risc. 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.

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 <i>forwarding</i> de dados e BTB
Processador #2	sem <i>forwarding</i> de dados, com NOPS e com BTB
Processador #3	com <i>forwarding</i> de dados e sem BTB
Processador #4	sem <i>forwarding</i> 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 [ $\mu$ s]
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 [ $\mu$ s]
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 [ $\mu$ s]
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.

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 se transfigurou num 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

 this sentence is weird . . . . .	1
 falar mais aqui . . . . .	1
 Imagem DetecaodeconflitoEXMEM . . . . .	3
 Imagem DetecaodeconflitoEXMEM2 . . . . .	3
 Imagem DetecaodeconflitoWB . . . . .	3
 imagem do xilinx do caminho critico - 183 . . . . .	3
 imagem do xilinx do caminho critico - 211 . . . . .	4