Universidade Federal de Minas Gerais – UFMG
Instituto de Ciências Exatas – ICEX
Departamento de Ciência da Computação – DCC
Trabalho Prático de OC2 – 2º Semestre Letivo de 2016
Professor: Omar Paranaiba Vilela Neto
Monitores: Mateus Tymburibá e Thiago Rodrigues

Especificação detalhada da 4ª entrega do Trabalho Prático

A 4ª etapa do trabalho prático consiste em implementar, na linguagem de descrição de hardware Verilog, todos os estágios da versão pipeline do microprocessador MIPS de 32 bits com suporte à inserção de bolhas (*stalls*) e encaminhamento. Essa implementação deve ser gravável em uma placa de FPGA DE2-115 (família Cyclone IV) ou DE2 (família Cyclone II) e suportar, no mínimo, as seguintes instruções:

- add (soma)
- addi (soma com imediato/constante)
- sub (subtração)
- lw (load word)
- sw (store word)
- and (and bit a bit)
- andi (and bit a bit com imediato/constante)
- or (or bit a bit)
- ori (or bit a bit com imediato/constante)
- nor (nor bit a bit)
- xor (xor bit a bit)
- slt (set if less than)
- slti (set if less than imediato/constante)
- sll (shift left logical)
- srl (shift right logical)
- beq (branch on equal)
- bne (branch on not equal)
- j (jump)

A fim de garantir a consistência das diversas implementações a serem elaboradas pelos grupos, abaixo estão reproduzidos alguns detalhes de projeto da versão pipeline do MIPS32 com suporte à inserção de bolhas (*stalls*) e encaminhamento, conforme disponível em [1].

Formato das instruções

As seguintes expressões serão usadas para indicar a finalidade dos campos das instruções:

- op: código de operação (*opcode*)
- rs: 1º registrador-fonte
- rt: 2º registrador-fonte
- rd: registrador-destino
- shamt: quantidade de bits a serem deslocados (*shift amount*)

- funct: código de função (indica operação específica da ULA)

Formato R:

op	rs	rt	rd	shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

- Instruções add, sub, and, or, nor, xor e slt:

Executam a operação indicada pelos campos “op” e “funct” da instrução sobre os registradores indicados nos campos “rs” e “rt”. Guardam o resultado no registrador indicado pelo campo “rd” da instrução.

- Instruções sll e srl:

Nessas instruções, o campo correspondente ao 2º operando (rt) deve ser desprezado (sem efeito). O número de bits do operando (rs) a serem deslocados é indicado no campo “shamt” em formato de número inteiro sem sinal.

Formato I:

op	rs	rt	constant or address
6 bits	5 bits	5 bits	16 bits

- Instruções lw e sw:

Calculam um endereço de memória somando o conteúdo do registrador indicado no campo “rs” ao valor imediato (constante/endereço) indicado na instrução (bits 15 a 0). Para isso, o valor imediato é expandido para 32 bits através da extensão de seu sinal (replicação do bit 15 nos bits mais significativos: bits 31 a 16).

lw: o valor lido da memória é gravado no registrador indicado pelo campo “rt” da instrução.

sw: o valor armazenado no registrador indicado pelo campo “rt” da instrução é gravado na memória.

- Instruções andi, ori, slti:

Executam a operação indicada pelos campos “op” e “funct” da instrução sobre o registrador indicado no campo “rs” e o valor imediato. Para isso, o valor imediato é expandido para 32 bits através da extensão de seu sinal (replicação do bit 15 nos bits mais significativos: bits 31 a 16). Essas instruções guardam o resultado no registrador indicado pelo campo “rt”.

- Instruções beq e bne:

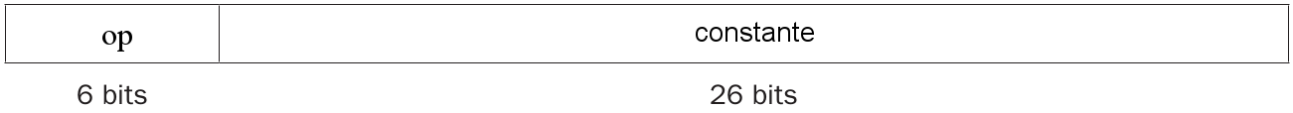
Comparam o conteúdo dos registradores indicados nos campos “rs” e “rt” da instrução. Para isso, efetuam uma subtração entre o conteúdo desses registradores. Se o resultado for igual a zero (registradores com conteúdo igual), o sinal de saída “Zero” da ULA é ativado (valor 1). Caso contrário, o sinal de saída “Zero” da ULA é desativado (valor

0). Esse sinal de saída é usado para definir se o desvio condicional deve ser efetuado ou não. Abaixo, na seção Caminho de Dados, é possível observar o uso de uma porta “and” para avaliar a condição de instruções “beq”. Os alunos deverão modificar esse caminho de dados para incluir suporte à instrução “bne”.

O endereço de destino do eventual desvio é calculado somando o valor imediato da instrução ao valor de PC + 1 (PC é o registrador Contador de Programa, *Program Counter*, que guarda o endereço da instrução a ser executada). Para isso, o valor imediato é expandido para 32 bits através da extensão de seu sinal (replicação do bit 15 nos bits mais significativos: bits 31 a 16).

OBS: como a memória especificada para este trabalho endereça apenas palavras de 4 bytes, ao contrário do MIPS que endereça cada byte da memória, o endereço de PC sempre deverá ser acrescido de 1 unidade em todas as instruções (ao invés de somar 4 unidades, como é mostrado no livro).

Formato J:



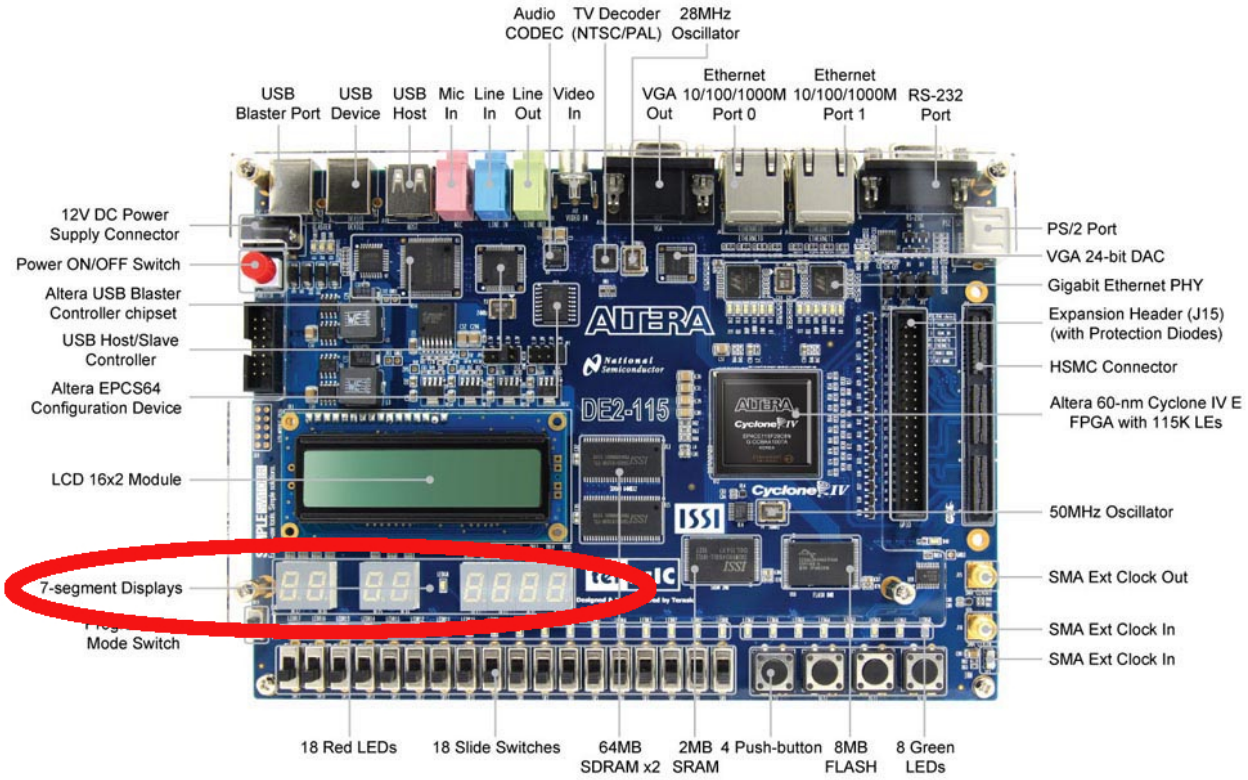
- Instrução j:

Executa um desvio incondicional concatenando a constante (valor imediato indicado na instrução) com os 6 bits mais significativos de PC + 1.

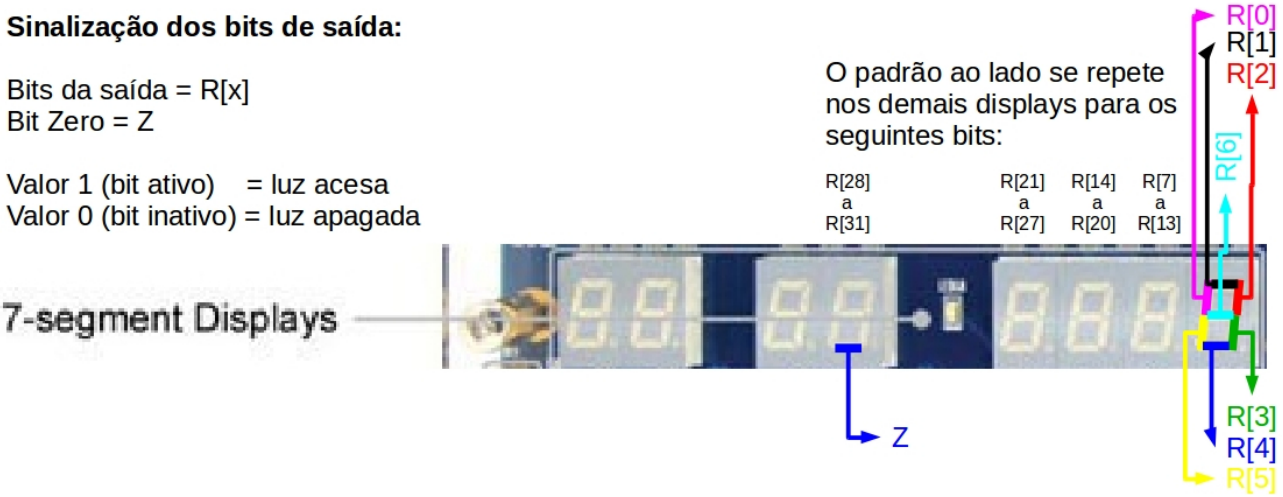
Análise dos resultados

Nesta etapa do trabalho, os valores armazenados nos registradores de destino (para instruções aritméticas e *loads*), as saídas da ULA (para instruções de desvio condicional e *stores*) e o valor de PC (para desvios incondicionais) serão utilizados para avaliar a correção da implementação. Portanto, as implementações em Verilog devem utilizar mecanismos de saída que informem ao avaliador os 32 bits do campo “resultado” da ULA, ou os 32 bits do registrador destino da instrução, ou os 32 bits do PC. Além disso, o valor do bit “zero” de saída da ULA sempre deve ser indicado.

Versões do código voltadas para simulação no ModelSim podem utilizar funções de impressão na tela como $monitor e $display para facilitar a visualização dos resultados. Quando utilizadas pelo grupo, essas versões devem ser submetidas para os avaliadores. Contudo, é obrigatória a criação de uma versão do código para execução nas placas FPGA. Nesse caso, as saídas mencionadas acima devem ser mapeadas para os displays numéricos de 7 segmentos existentes na placa, da seguinte maneira:



Indicação da localização na placa FPGA dos displays numéricos de 7 segmentos.



Padronização de exibição das saídas nos displays.

Alternativamente, os valores de saída podem ser exibidos em formato hexadecimal nos displays, ao invés do formato indicado acima.

A escolha do valor a ser exibido nos displays deve ser feita através dos 2 “slide switches” mais à direita da placa FPGA, da seguinte forma (0 = switch para baixo e 1 = switch para cima):

- 00 → exibir o valor armazenado no registrador destino da instrução.
- 01 → exibir o valor de saída da ULA.
- 10 → exibir o valor de PC.

Os alunos deverão criar conjuntos de testes para verificar o funcionamento de cada uma das instruções em diversos cenários. Esse conjunto de testes elaborado pelos alunos também deve ser encaminhado aos avaliadores. Contudo, os avaliadores utilizarão um conjunto de testes próprio para confirmar o correto funcionamento do código. Cuidem para gerar exemplos de testes que contenham conflitos de dados que exijam a inserção de bolhas e que exijam o encaminhamento de dados.

Além disso, cada grupo deverá entregar uma versão modificada do caminho de dados exibido nesta especificação. O objetivo é adaptá-lo para oferecer suporte a todas as instruções elencadas neste enunciado.

Todo o material a ser entregue deve ser submetido por e-mail, até o fim do dia 09/12/2016, para os seguintes endereços de e-mail:

- omar@dcc.ufmg.br
- mateustymbu@dcc.ufmg.br
- thiagorbss@gmail.com

O material a ser anexado ao e-mail deverá ser reunido em uma pasta com o seguinte nome (OBS: o caractere X abaixo deve ser substituído pelo número do grupo. Os números dos grupos estão indicados no Apêndice A deste enunciado):

TP-OC2_Entrega3_GrupoX

Em seguida, essa pasta deverá ser compactada, gerando um arquivo com o seguinte nome:

TP-OC2_Entrega3_GrupoX.zip

Certifique-se de que o arquivo não está corrompido.

O título do e-mail a ser enviado deverá ser:

TP-OC2_Entrega3_GrupoX

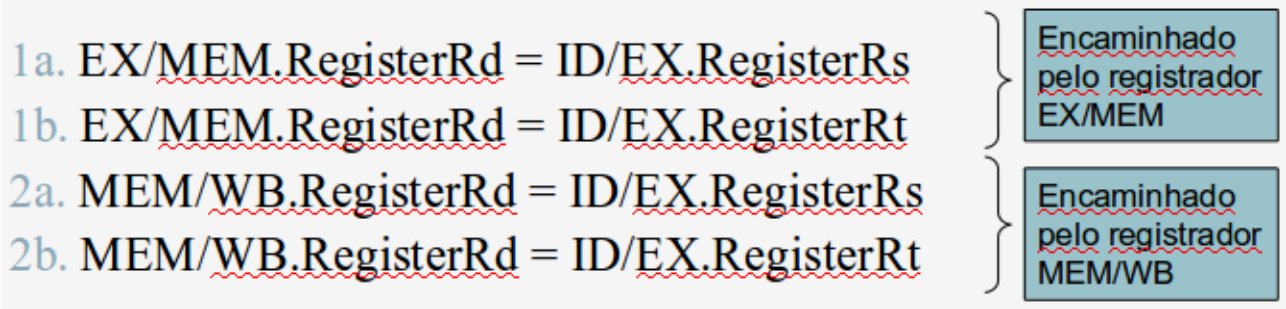
Siga rigorosamente o padrão de nomes descrito acima.

A fim de evitar problemas na abertura de arquivos, certifique-se de que nenhum arquivo incluído na pasta a ser compactada possui caracteres especiais em seus nomes (cedilha, acentos, espaços, etc).

Os integrantes de cada grupo serão avaliados individualmente. Para isso, deverão comparecer, juntos, a uma entrevista sobre o trabalho a ser agendada posteriormente. Nessa ocasião, todos os integrantes deverão responder perguntas sobre as decisões de projeto e de implementação adotadas pelo grupo, além de demonstrar o funcionamento do código.

Encaminhamento

Para ser capaz de detectar a necessidade de encaminhamento, o processador deve passar adiante os números dos registradores pelo pipeline (Rs, Rt e Rd). Por exemplo, conflitos ocorrem quando:



Além disso, antes de efetuar o encaminhamento, o processador deve se certificar de que as seguintes condições são satisfeitas:

- Apenas a instrução encaminhadora vai escrever em registrador:

  EX/MEM.RegWrite == 1
  MEM/WB.RegWrite == 1

- Apenas se o Rd (destino) não for $zero:

  EX/MEM.RegisterRd ≠ 0
  MEM/WB.RegisterRd ≠ 0

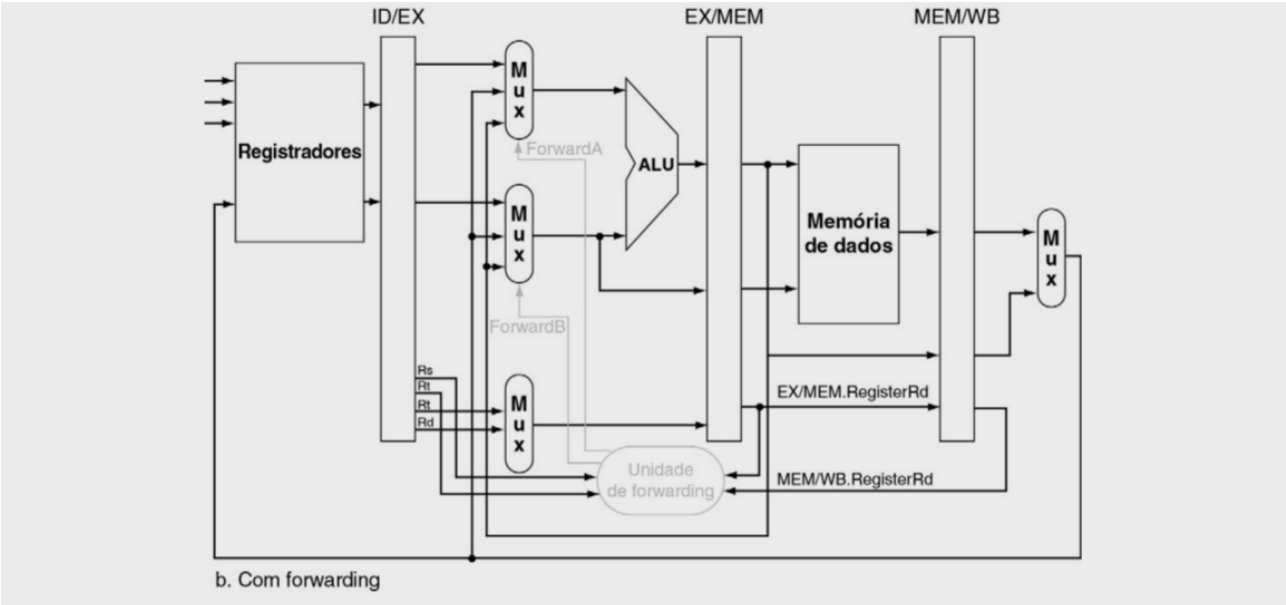


FIGURA 4.54 Em cima estão a ALU e os registradores de pipeline antes da inclusão do forwarding. Embaixo, os multiplexadores foram expandidos para acrescentar os caminhos de forwarding, e mostramos a unidade de forwarding. O hardware novo aparece em um destaque. No entanto, essa figura é um desenho estilizado, omitindo os detalhes do caminho de dados completo, como o hardware de extensão de sinal. Observe que o campo ID/EX.RegistradorRt aparece duas vezes, uma para conectar ao mux e uma para a unidade de forwarding, mas esse é um único sinal. Como na discussão anterior, isso ignora o forwarding de um valor armazenado por uma instrução store. Observe que esse mecanismo também funciona para instruções `slt`.

Portanto, as condições para detecção de conflitos podem ser descritas da seguinte forma:

Conflito entre EX e MEM

```
if ( (EX/MEM.RegWrite = 1)    and
     (EX/MEM.RegisterRd ≠ 0) and
     (EX/MEM.RegisterRd = ID/EX.RegisterRs) )

  ForwardA = 10

if ( (EX/MEM.RegWrite = 1)    and
     (EX/MEM.RegisterRd ≠ 0) and
     (EX/MEM.RegisterRd = ID/EX.RegisterRt) )

  ForwardB = 10
```

Conflito entre EX e WB

```
if ( (MEM/WB.RegWrite)          and
     (MEM/WB.RegisterRd ≠ 0) and
     not ( (EX/MEM.RegWrite = 1) and
           (EX/MEM.RegisterRd ≠ 0) and
           (EX/MEM.RegisterRd = ID/EX.RegisterRs) ) and
     (MEM/WB.RegisterRd = ID/EX.RegisterRs) )

  ForwardA = 01

if ( (MEM/WB.RegWrite= 1)    and
     (MEM/WB.RegisterRd ≠ 0) and
     not ( (EX/MEM.RegWrite = 1) and
           (EX/MEM.RegisterRd ≠ 0) and
           (EX/MEM.RegisterRd = ID/EX.RegisterRt) ) and
     (MEM/WB.RegisterRd = ID/EX.RegisterRt) )

  ForwardB = 01
```

As condições em vermelho tratam casos de conflito duplo (EX com WB e EX com MEM), situação em que deve-se usar o dado mais recente (oriundo de EX/MEM).

Inserção de bolhas (*stalls* ou paradas)

Conflitos do tipo Load/Use podem ser detectados através das seguintes condições:

```
if ( (ID/EX.MemRead = 1) and
     ( (ID/EX.RegisterRt = IF/ID.RegisterRs) or
       (ID/EX.RegisterRt = IF/ID.RegisterRt) ) )
```

Nesse caso, deve ser inserida uma bolha. Para inserir tal bolha, deve-se:

- Forçar os sinais de controle em ID/EX para 0. O efeito disso é fazer com que os estágios EX, MEM e WB da instrução executem uma instrução nop (no-operation).
- Prevenir a atualização do PC e do IF/ID. Com isso, a instrução parada (Uso) é decodificada novamente e a instrução subsequente é buscada novamente.

Caminho de dados (*data path*) do MIPS para encaminhamento e paradas

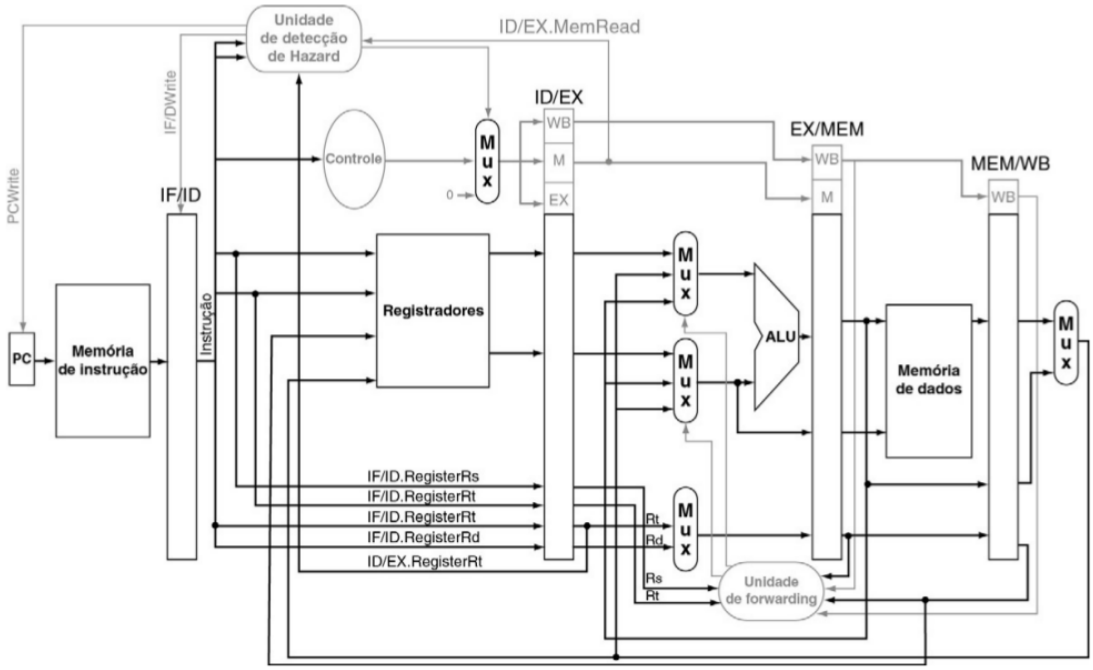


FIGURA 4.60 Visão geral do controle em pipeline, mostrando os dois multiplexadores para forwarding, a unidade de detecção de hazard e a unidade de forwarding. Embora os estágios ID e EX tenham sido simplificados – a lógica de extensão de sinal imediato e de desvio estão faltando –, este desenho mostra a essência dos requisitos do hardware de forwarding.

Observações:

- Algumas instruções, como “sll”, “slr” e “bne”, poderão exigir ajustes no *data path* apresentado. A constante 4 deve ser substituída pela constante 1.
- 1 ciclo de parada é suficiente para MEM ler o dado no lw, pois, em seguida, já é possível encaminhar o dado para o estágio EX da instrução dependente.
- Conflitos de controle (decorrentes de instruções de desvios) podem exigir o descarte de instruções posicionadas no primeiro estágio. Uma forma de tratar isso é criar um sinal de escrita no registrador intermediário (IF/ID), usado para transformar a instrução buscada em uma instrução nop (no-operation).
- O livro mostra a inclusão de hardware no estágio ID para adiantar o cálculo de endereços de desvios e de condições de desvios condicionais. Nesse caso, o tratamento de conflitos de dados (encaminhamentos ou bolhas) entre outras instruções e instruções de desvio deve levar em consideração que os dados usados por instruções de desvio precisam estar disponíveis no início do estágio ID.

Referência

[1] Patterson, D. Hennessy, J. “Organização e Projeto de computadores: a interface Hardware/Software”. 4ª edição. Seções 4.7 e 4.8.

Apêndice A – Listagem atualizada dos grupo

Nº do grupo	Integrantes
1	Sebastião Mendes
	Luis Pedraza
	Tiago Amador
	Gabriel Noreg
	Mateus Rezende
2	Gabriel Carvalho
	Juliana Ramos
	Lucas Machado
	Marcel Henrique
	Nélio Cézar
3	Adler Melgaço Ferreira
	João Paulo Bregunci
	Marina Monteiro Moreira
	Pedro Elias Valadares Castanheiras
	Ronald Davi Rodrigues Pereira
4	Giovanni Leite
	Rafael Rubiolli
	Fernanda Ramalho
	Danilo Viana
	Manoel Junior
5	Delisson Junio
	Gabriel Oliveira
	Lucas Peixoto
	Pedro Paulo
	Rafael Grandrire
6	Jota Vicente
	Pedro Dalla
	Luiz Otávio
	Ivan Soares
	Edson Roteia
7	David Alexandre
	Diogo Leite
	Matheus Filipe Sieiro Vargas
	Matheus de Paula
8	Jéssica Cristina Carneiro
	Luiz Carlos de Oliveira
	Nathalia Campos
	Fabio Lelis
9	Andrei dos Santos Silva
	Michael Lopes
	Marlon Dias
	Ivanei Souza
	Julio Leandro
10	Dourival Pimentel
	Ana Luiza de Avelar