UNIVERSIDADE DE BRASÍLIA
DEPARTAMENTO DE CIÊNCIAS DA COMPUTAÇÃO
ORGANIZAÇÃO E ARQUITETURA DE COMPUTADORES - 1º/2016

Laboratório 5 - CPU MIPS Pipeline

Alunos:

André Abreu R. de Almeida - 12/0007100
Arthur de Matos Beggs - 12/0111098
Bruno Takashi Tengan - 12/0167263
Gabriel Pires Iduarte - 13/0142166
Guilherme Caetano - 13/0112925
João Pedro Franch - 12/0060795
Rafael Lima - 10/0131093

1) Analise o processador MIPS PipelineM fornecido. Desenhe o Diagrama de Blocos do Caminho de Dados completo incluindo os registradores de pipeline e especifique a tabela verdade dos sinais de controle por estágio do pipeline:

2) Analise as unidades de HazardM e ForwardM, e com base na ISA especifique, através de exemplos, quais riscos de dados e de controle são detectados e tratados:

A unidade de forward intervirá quando:

1. A instrução no estagio MEM necessitar escrever no banco dos registradores e o registrador Rd, não sendo o registrador $zero, de tal instrução for igual ao registrador Rs ou Rt da instrução no estagio EX.
2. Quando a condição anterior não for satisfeita, pois forward relacionado ao estagio MEM tem mais prioridade por ser mais recente, e a instrução no estagio WB necessitar escrever no banco dos registradores. Além do registrador Rd, não sendo o registrador $zero, de tal instrução em WB for igual ao registrador Rs ou Rt da instrução no estagio EX.
3. A instrução no estagio MEM necessitar escrever no banco dos registradores e o registrador Rd, não sendo o registrador $zero, de tal instrução for igual ao registrador Rs ou Rt da instrução no estagio ID.

Dessa forma as condições 1 e 2, farão a unidade de forward selecionar as entradas adequadas nos MUX de entradas A e B da ULA. Já a condição 3 efetuará nos MUX para das entradas no calculo de branch no estagio ID.

Assim um exemplo, satisfazendo a condição 1, será necessario que a unidade de forward realize um tratamento para risco de dados é dado abaixo.

```
add $s0, $t1, $t2
add $s1, $s0, $s0
```

Exemplo 1 - Satisfaz condição 1 da unidade de forward.

Assim um exemplo, satisfazendo a condição 2, será necessario que a unidade de forward realize um tratamento para risco de dados é dado abaixo.

```
lw $s0, 0($t1)
add $t2, $t3, $t4
add $s1, $s0, $s0
```

Exemplo 2 - Satisfaz condição 2 da unidade de forward.

Assim um exemplo, satisfazendo a condição 3, será necessario que a unidade de forward realize um tratamento para risco de dados é dado abaixo.

```
add $s0, $t1, $t2
add $t2, $t3, $t4
add $s1, $s0, $s0
```

Exemplo 3 - Satisfaz condição 3 da unidade de forward.

A unidade de Hazard operará quando:

I. O registrador Rd da instrução no estagio EX for o mesmo registrador seja Rs ou Rt da instrução no estagio ID.

II. O registrador Rd da instrução no estagio MEM for o mesmo registrador seja Rs ou Rt da instrução no estagio ID.

III. A instrução no estagio EX for ler da memória quando estiver no estagio WB ou a instrução no estagio ID for um branch, beq ou bne, e a instrução em EX vá escrever no banco de registradores, sendo o registrador Rs diferente de $zero, juntamente como a condição 1 sendo verdadeira

IV. A instrução no estagio ID for um branch, beq ou bne, e a instrução em MEM for escrever no banco de registradores e ler da memória no estagio WB, sendo o registrador Rs diferente de $zero e condição 2 verdadeira.

V. A instrução em ID for um jr e a instrução em EX tiver o registrador destino igual ao registrador Rs da instrução em ID.

VI. A instrução em ID for jr e a instrução em MEM tiver o registrador destino igual a $ra.

Sendo assim se a condição III ou IV for satisfeita se criará um pipeline stall, conhecido como bolha. A condição V selecionará o correto valor do registrador como origem para a origem de PC caso a instrução seja um *jr*. Caso entrasse na condição VI deveria ser realizado um tratamento como na condição V, porém não foi encontrado no caminho de dados tal correção.

```
lw $s0, 0($t1)
add $s1, $s0, $s0
```

Exemplo 4 - Satisfaz condição III da unidade de Hazard.

```
beq $s0, $s0, label1
add $s1, $s2, $s3
```

Exemplo 5 - Satisfaz condição III da unidade de Hazard.

```
add $ra, $s2, $s3
jr $ra
```

Exemplo 6 - Satisfaz condição IV da unidade de Hazard.

```
add $ra, $s2, $s3
add $s1, $s2, $s3
jr $ra
```

Exemplo 7 - Satisfaz condição V da unidade de Hazard.

3) Use o seu programa teste.s e verifique o correto funcionamento de TODAS as instruções da ISA implementada, teste usando simulação por forma de onda e pela implementação na DE2:

O arquivo de testes “teste.s” se encontra na pasta Lab5.

4) Encontre a frequência máxima de clock do processador na qual a ISA ainda é corretamente executada:

Para determinar a frequência máxima de operação do processador, deve-se analisar quanto tempo o processador leva para executar a etapa mais demorada, que de acordo com a base teórica seria o acesso à memória de dados. Para a verificação experimental, utilizamos nosso arquivo teste.s com todas as funções implementadas e verificamos o seu funcionamento para diferentes frequências de clock. Utilizando como base o divisor implementado nas SW[7..0] da FPGA DE2-70, viu-se que para a disposição das switches SW[7..0]=01000110 o processador não executava mais todas

as instruções corretamente. O valor obtido da disposição das switches é de 70, logo a frequência máxima de operação é de 50MHz/70, que é aproximadamente 700kHz.

5) Implemente as instruções abaixo em conformidade com a ISA MIPS (livro See MIPS Run e Manual do MIPS):

bgez $t0,LABEL # $t0>=0 ? PC=LABEL : PC=PC+4

bgezal $t0,LABEL # $t0>=0 ? PC=LABEL e $ra=PC+4 : PC=PC+4

bgtz $t0,LABEL # $t0>0 ? PC=LABEL : PC=PC+4

blez $t0,LABEL # $t0<=0 ? PC=LABEL : PC=PC+4

bltz $t0,LABEL # $t0<0 ? PC=LABEL : PC=PC+4

bltzal $t0,LABEL # $t0<0 ? PC=LABEL e $ra=PC+4 : PC=PC+4

a e b) Indique as modificações necessárias no caminho de dados. / Indique as modificações necessárias no bloco de controle:

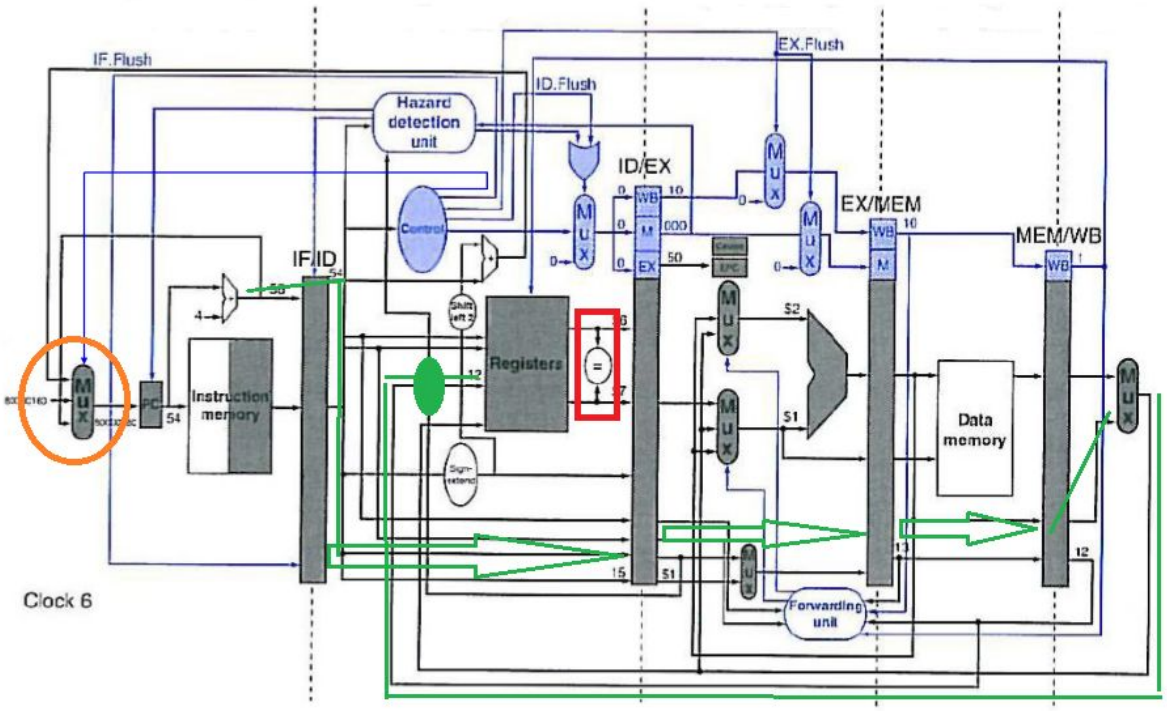

Figura 1 - Alterações e pontos de destaque no Pipeline

A figura acima demonstra o pipeline, marcados os pontos de destaque para a implementação dos Branchs. Em vermeho está a unidade de comparação para tomada de BEQs. Ela foi substituída em nossa versão do pipeline por uma unidade de processamento de branchs. Essa estrutura se encontra no arquiivo \MIPS_PUMv4.3\Core\CPU\Branch_PIPEM.v.

O Branch_PIPEM se encarrega de analizar o opcode diretamente da instrução lida, e gerar sinais de controle de tomada de branch ou nao (que por sua vez ativa o flush caso o branch seja tomado). O multiplexador marcado em laranja, o OrigALU passa a ser controlada pela estrutura de branchs, no caso onde um branch ocorre.

Essa estrutura tem como entradas também o campo $rt, e as tradicionais entradas iA e iB da ula que podem ser conectadas em qualquer entrada de 32 bits. Para o pipeline elas são conectadas aos campos $rs e $rt (o que torna uma entrada redundante, ela pode ser extraída no futuro) para a comparação de BEQ e BNE.

Com iA e $rt, os demais branchs são implementados. As comparações são feitas entre iA e o parâmetro ‘ZERO’, que são 32’b0. A operação efetuada é definida pelo conjunto de OPCODE e campo $rt, este funcionando como um equivalente ao ‘funct’ das operações tipo R.

Além da saída ‘oBranch’, que define se o branch é tomado ou não, está presente também uma saída ‘oLink’, que define se o registrador $ra deve ser atualizado com o valor de PC+4 ou não. Essa saída controla também o multiplexador RegDST, marcado

no diagrama da figura como um disco sólido em cor verde. As setas e o traço verdes, da mesma cor, definem o caminho feito pelo valor de PC+4 até a sua gravação em $ra.

c) Crie um programa teste que comprove o correto funcionamento das novas instruções:

O programa testeBRANCH.s está anexo.

6) Implemente a chamada do sistema Syscall 49, que receba como argumentos: $a0 o endereço do cartão SD, $a1 endereço da memória e $a2 a quantidade de bytes a serem transferidos, e retorne $v0=0 em caso de sucesso e $v0=1 em caso de erro. Documente a implementação no arquivo Mips.docx:

O controlador do cartão SD está 100% funcional, o syscall 49 implementado e funcionando conforme esperado e o arquivo de testes “testeReadSD.s” presente em “Docs > testes” exemplifica o uso do syscall 49.

A implementação do controlador foi documentada no arquivo “Mips.docx”.

Observação: O controlador já estava funcional na entrega do Lab4, porém a leitura estava sendo feita a partir do endereço lógico do cartão, e não do endereço físico. Há um offset de 137 setores (cada setor de 512 bytes) entre os dois endereços, e o dado retornado do endereço “x” não correspondia ao esperado pois era na verdade o dado do endereço “x + 0x00011200”.