Hazards

Yuri Kaszubowski Lopes

UDESC

Anotações

Hazards

- Pipeline de 5 estágios: IF, ID, EX, MEM, WB.
- Ao implementarmos o conceito de pipeline, criamos diversos problemas e complexidades
  - Dentre os problemas, estão os hazards
  - Hazard: risco ou perigo
  - Impedem que a próxima instrução continue sua execução no ciclo de clock seguinte
- Tipos:
  - Hazards Estruturais
  - Hazards de Dados
  - Hazards de Controle

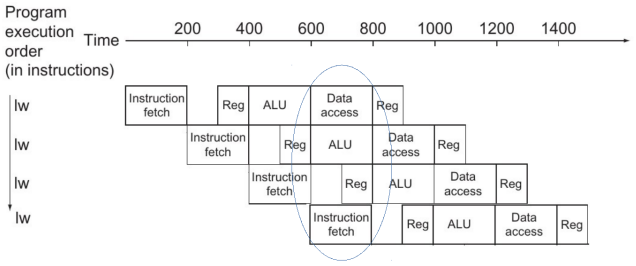
Anotações

Hazards Estruturais

- Em um hazard estrutural o hardware não pode manter duas instruções no pipeline, pois elas estão competindo por algum componente
- e.g., se tivéssemos uma única memória, para dados e instruções:
  - Enquanto uma instrução está sendo carregada, outra está tentando escreverna memória ao mesmo tempo
  - Se tivéssemos uma única memória (para instruções e dados), em algum momento no pipeline duas instruções poderiam competir pelo acesso a essa memória
- e.g., se nosso banco de registradores não permitisse ler e escrever ao mesmo tempo:
  - Não podemos ter ID e WB ao mesmo tempo

Anotações

Hazards Estruturais



- Nossa implementação do MIPS não sofre com hazards estruturais
  - Os componentes que poderiam conflitar em determinado instante já estão duplicados (e.g., memórias separadas, banco de registradores)
  - Um hazard estrutural muitas vezes é corrigido criando-se cópias das unidades funcionais, para instruções em diferentes estágios do pipeline utilizem diferentes cópias da unidade funcional

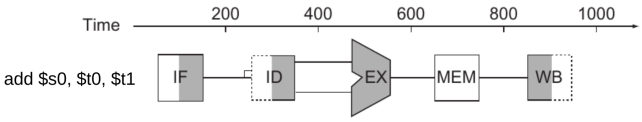
Anotações

Hazards de Dados

- Considere as seguintes instruções

```
1 add $s0, $t0, $t1
2 sub $t2, $s0, $t3
```

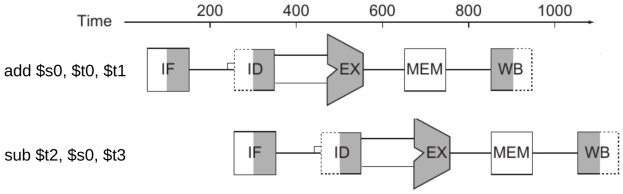
- Considere diagrama da execução do add, e adicione o diagrama do sub logo abaixo (em um pipeline). Identifique o que está sendo feito em cada etapa e qual o problema que está acontecendo.



- Notação:
  - Sombreado: indica que o elemento está sendo utilizado pela instrução
    - ... à direita: lido, à esquerda: escrito

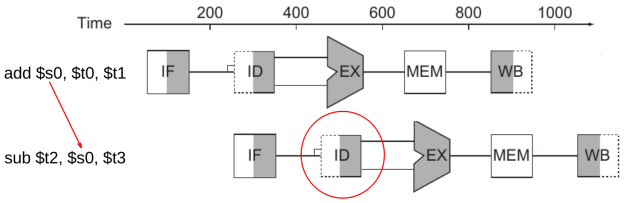
Anotações

Hazards de Dados



Anotações

Hazards de Dados



- O resultado da operação é escrito em `$s0` somente no estágio write-back (está pronto no picossegundo 1000)
- Os valores dos registradores estão sendo buscados em `ID`, e `$s0` ainda não está "pronto"
- O pipeline precisa ser interrompido pois uma etapa precisa que outra seja concluída para que o dado esteja pronto
  - Uma instrução depende de outra
  - No circuito, o resultado é escrito no estágio `WB`
    - Mas em que estágio o resultado já está "pronto" e só não foi gravado?
    - Como utilizar isso para resolver/minimizar o problema?

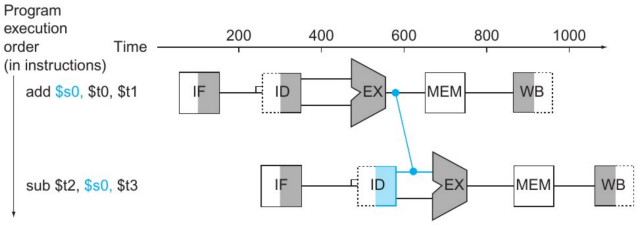
Anotações

Forwarding

- Alguns hazards de dados podem ser mitigados por **forwardings**
  - Conhecidos também como **bypassings**
  - "Emprestamos" o resultado de uma unidade antes da operação ter sido completada (escrito na memória ou banco de registradores)
- Veja que o resultado já está pronto após a unidade EX, ele só não foi escrito no registrador ainda
  - Podemos tomar esse resultado, e utilizar como entrada para o EX da próxima instrução
  - Vai exigir um controle mais complexo da nossa CPU
    - Agora devemos decidir ainda se a entrada da ALU vêm do banco de registradores ou do resultado atual da ALU

Anotações

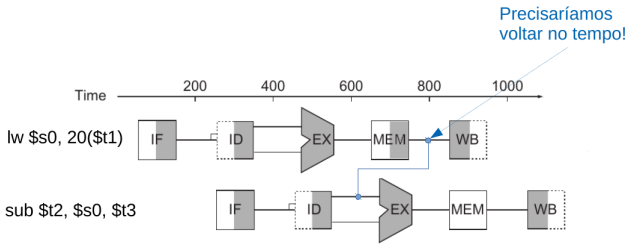
Forwarding



Anotações

Exercício

1. Faça uma sequência de instruções onde o forwarding não vai resolver o problema



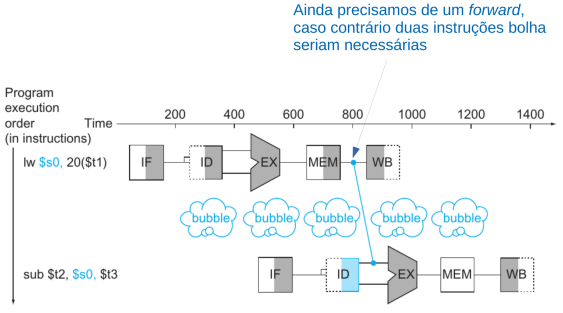
Anotações

Pipeline Stall (Bolhas)

- Um hazard de dados pode exigir o atraso da execução da próxima instrução
- Ao detectar isso, o processador deve injetar uma **instrução bolha** entre as instruções que estão com o hazard
  - A instrução bolha não faz operação alguma
  - Não altera o estado dos registradores e não altera a memória de dados
  - Somente gasta tempo, para que o resultado necessário possa ser computado
- Geramos um pipeline stall
  - Tivemos que parar nossa “linha de montagem” por um momento para então continuar

Anotações

Pipeline Stall



- Usamos **um** stall e forward
- Sem forward, seria necessário pelo menos **dois** stalls:
  - Considerando que com WB e ID no mesmo clock, WB termina antes de ID “pegar” os valores no registrador

Anotações

O papel do compilador

- No programa abaixo, o que o compilador ou programador poderiam fazer para tentar evitar um stall?

```
1 addi $t5, $t5, 10
2 lw $s0, 20($t1)
3 sub $t2, $s0, $t3
```

- O compilador pode reordenar as instruções ao gerar o código de máquina (ou assembly), e nesse caso isso não afetaria o resultado do programa, mas evitaria um stall

```
1 lw $s0, 20($t1)
2 addi $t5, $t5, 10
3 sub $t2, $s0, $t3
```

Anotações

O papel do compilador

- Compiladores sofisticados, como o GCC, tentam de todo modo criar uma ordem nas instruções de modo a evitar stalls no pipeline
  - A CPU pode por conta própria executar as instruções em uma ordem diferente para evitar stalls
  - Utiliza técnicas como buffers de reordenação e renomeação de registradores
    - Complexo! Não cobriremos isso em nossa disciplina introdutória.
    - Uma descrição (básica) sobre como fazer isso pode ser encontrada em Henessy; Patterson. Computer Architecture: A Quantitative Approach. 6a Edição. 2017.

Anotações

Hazards de Controle

- O pipeline funciona encadeando as instruções, uma após a outra, em nossa “linha de montagem”
- Mas e se não sabemos qual é a próxima instrução?
  - Em que cenário isso pode acontecer?

```
1 add $4, $5, $6
2 beq $1, $2, 1
3 lw 3, 300($0)
4 or $7, $8, $9
```

- Até o `beq` terminar o estágio de `EX` não sabemos se executamos o `lw`

Anotações

Hazards de Controle

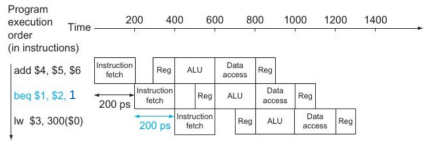
Solução “simples”

- Sempre considerar que o desvio não foi tomado
- Quando a instrução de desvio terminar de executar:
  - Se o desvio realmente não devia ser tomado, tudo continua normalmente
  - Se descobrirmos que o desvio deveria ser tomado, descartamos as instruções que estão no pipeline e carregamos as instruções a partir do endereço correto
    - Injetar bolhas no meio do caminho!

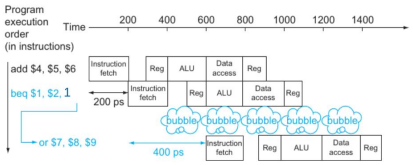
Anotações

Hazards de Controle

```
1 add $4, $5, $6
2 beq $1, $2, 1
3 lw 3, 300($0)
4 or $7, $8, $9
```



Se a CPU concluir que o desvio deveria ser tomado, precisamos descartar as etapas do lw processadas, trocar por bolhas, e colocar a instrução correta na fila do pipeline.



Anotações

Hazards de Controle

- Podemos ainda adicionar unidade de predição dinâmica de desvios
  - A CPU tenta “adivinhar” se o desvio vai ser tomado ou não
  - e.g., mantemos uma tabela com o endereço da instrução, e se o desvio foi tomado na última vez que executamos essa instrução ou não
    - Especialmente útil para desvios em loops
    - Essas técnicas, aliadas a máquinas de estados e outras mais, são as usadas nas CPUs atuais
    - Segundo Patterson e Henessy (2014), a taxa de acertos dessas técnicas chega a 90%

Anotações

Implementando

- Já sabemos
  - O que é um pipeline
  - Os hazards que surgem com o pipeline
- Precisamos implementar o pipeline na nossa CPU!

Anotações

Exercícios

2. Foi demonstrado que uma das entradas da unidade `EX` pode vir emprestada da etapa `EX` anterior para as instruções:

```
1 add $s0, $t0, $t1
2 sub $t2, $s0, $t3
```

   Existe a possibilidade das duas entradas `EX` precisarem de um forwarding? Se sim, faça um código em Assembly que ilustre esse cenário, e desenhe o gráfico com os forwardings

Anotações

Exercícios

3. Para as sequências a seguir, indique se acontecerá um stall, se stalls podem ser evitados via forwarding, ou se a execução não gera stalls e não requer forwardings. Respostas no livro base da disciplina.
   1. a)

```
1 lw $t0,0($t0)
2 add $t1,$t0,$t0
```

   2. b)

```
1 add $t1,$t0,$t0
2 addi $t2,$t0,5
3 addi $t4,$t1,5
```

   3. c)

```
1 addi $t1,$t0,1
2 addi $t2,$t0,2
3 addi $t3,$t0,2
4 addi $t3,$t0,4
5 addi $t5,$t0,5
```

Anotações

Referências

- D. Patterson; J. Henessy. **Organização e Projeto de Computadores: Interface Hardware/Software**. 5a Edição. Elsevier Brasil, 2017.
- Andrew S. Tanenbaum. **Organização estruturada de computadores**. 5. ed. São Paulo: Pearson, 2007.
- Harris, D. and Harris, S. **Digital Design and Computer Architecture**. 2a ed. 2012.
- courses.missouristate.edu/KenVollmar/mars/

Anotações

Anotações

Anotações