Hazards de Dados: Construindo Forwardings (bypasses)

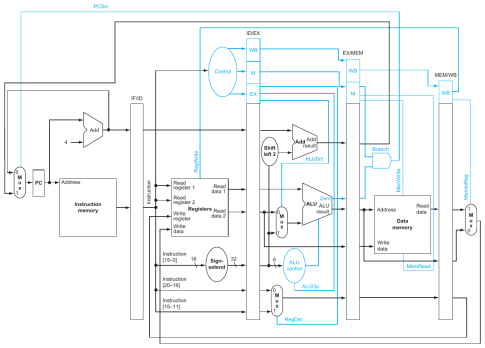
Yuri Kaszubowski Lopes

UDESC

Anotações

Hazards de Dados

- O pipeline construído até o momento funciona caso nossas instruções não tenham dependências
  - Mas vimos que hazards de **dados** e de **controle** são comuns
  - Vamos começar pelo tratamento dos hazards de dados: **Forwardings**



Anotações

Exemplo

- Considere as instruções a seguir

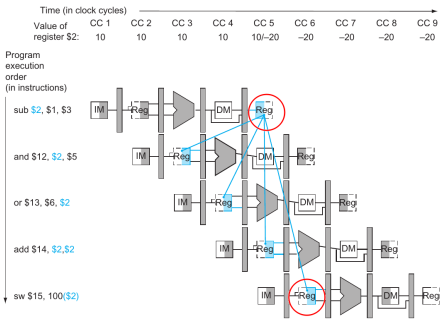
```
1 sub $2, $1, $3
2 and $12, $2, $5
3 or $13, $6, $2
4 add $14, $2, $2
5 sw $15, 100($2)
```

- Onde estão os hazards de dados nessas instruções?
  - Todas instruções marcadas dependem do resultado do sub
  - Se isso vai causar hazards de dados ou não depende diretamente de como nosso pipeline é montado, e de sua profundidade

Anotações

Exemplo: Pipeline de 5 estágios

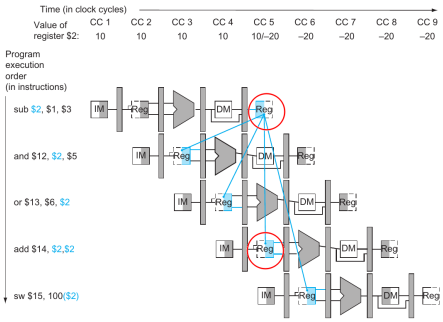
- `sub` armazena o resultado de `$2` no banco de registradores no clock 5
- `sw` está lendo `$2` no clock 6, então não temos hazard



Anotações

Exemplo: Pipeline de 5 estágios

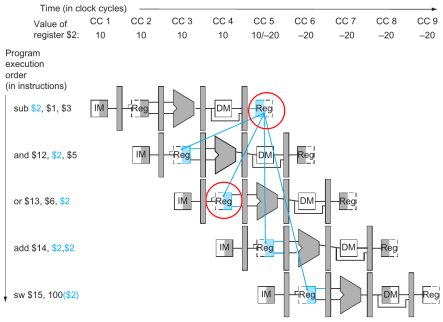
- `sub` armazena o resultado de `$2` no banco de registradores no clock 5
- `add` está lendo `$2` no mesmo clock que o resultado está sendo escrito. Isso não gera um hazard no banco de registradores. Os flip-flops são construídos de forma que o dado é escrito no início do ciclo, e são lidos no final do ciclo



Anotações

Exemplo: Pipeline de 5 estágios

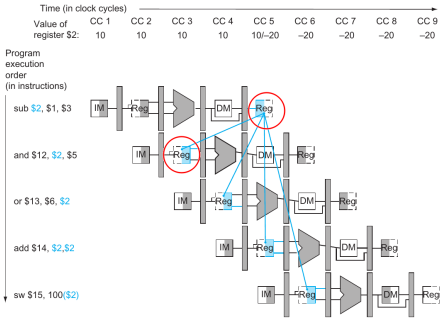
- `sub` armazena o resultado de `$2` no banco de registradores no clock 5
- `or` tenta usar `$2` antes do resultado estar pronto (gravado no banco de registradores). Temos hazard de dados aqui



Anotações

Exemplo: Pipeline de 5 estágios

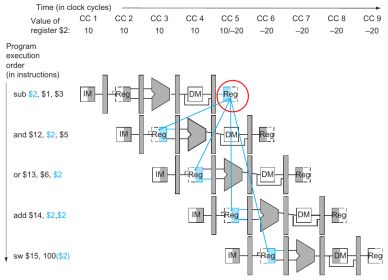
- `sub` armazena o resultado de `$2` no banco de registradores no clock 5
- `and` tenta usar `$2` antes do resultado estar pronto (gravado no banco de registradores). Temos hazard de dados aqui



Anotações

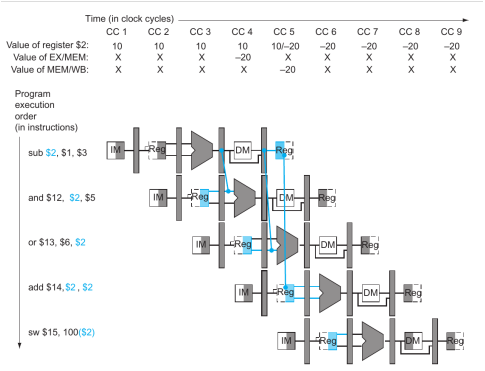
Exemplo: Pipeline de 5 estágios

- Em que estágio do `sub` (primeira instrução) o resultado já está pronto e só não foi gravado?
- E em que estágio esses valores são realmente utilizados pelas instruções subsequentes `and` e `or`?
  - O resultado está pronto quando sai da ALU
  - O valor é utilizado pela ALU nas demais instruções. E é aqui que precisamos fazer um forward.



Anotações

Exemplo: Forwardings

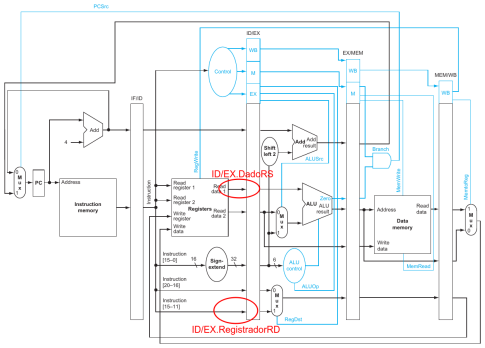


- Para o `add`, o forward já é feito “automaticamente” pelo banco de registradores

Anotações

Registradores de Pipeline

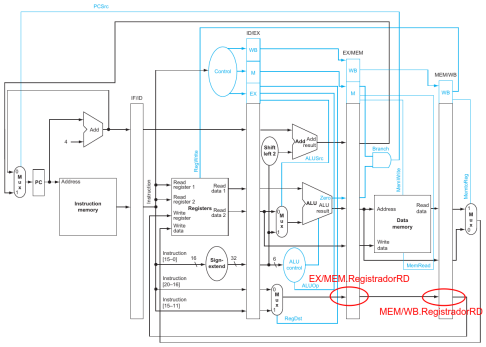
- Registradores de pipeline armazenam informações relevantes da instrução a cada estágio
- Nomes dos registradores de fronteira (pipeline): `IF/ID`, `ID/EX`, `EX/MEM`, `MEM/WB`
- O **dado do registrador fonte 1**, armazenado em **ID/EX**: `ID/EX.DadoRS`
- O **endereço do registrador destino**, armazenado em **ID/EX**: `ID/EX.RegistradorRD`



Anotações

Registradores de Pipeline

- No estágio `EX`, quando a ALU calcula um resultado, ele é armazenado no estágio `EX/MEM`
- Também é salvo o **endereço do registrador de destino**: `EX/MEM.RegistradorRD`
- O mesmo ocorre quanto a instrução passa pelo estágio `MEM`. O **endereço do registrador de destino** é salvo em: `MEM/WB.RegistradorRD`



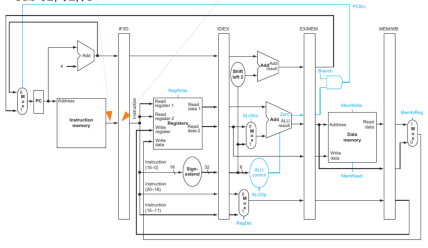
Anotações

Passo a passo de exemplo

Os 32 bits da instrução são lidos da memória de instruções e armazenados nos registradores de pipeline IF/ID

and $12,$2,$5 sub $2, $1,$3

Não entrou no pipeline

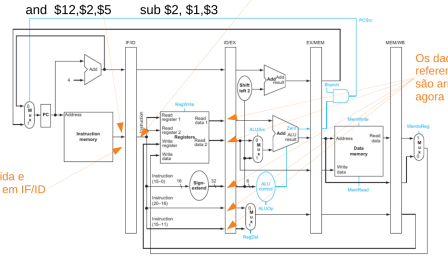


Anotações

Passo a passo de exemplo

A instrução salva anteriormente é carregada para continuar

… and $12,$2,$5 sub $2, $1,$3



Os dados referentes ao sub são armazenados agora em ID/EX

A próxima instrução é lida e enviada para ser salva em IF/ID

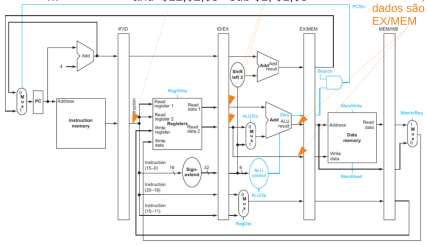
Anotações

Passo a passo de exemplo

Os dados são carregados no próximo estágio

Resultado (que poderá ser salvo no registrador destino rd) e os demais dados são salvos em EX/MEM

… … and $12,$2,$5 sub $2, $1,$3

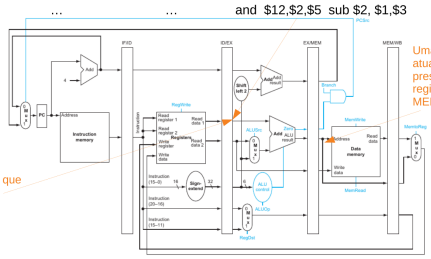


Anotações

Passo a passo de exemplo

Aqui as coisas dão errado!

… … … and $12,$2,$5 sub $2, $1,$3



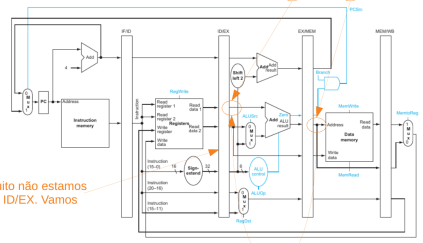
Uma versão mais atual de $2 está presente nos registradores EX/MEM

O and está pegando $2 que foi salvo anteriormente

Anotações

Forwarding

and $12,$2,$5 sub $2, $1,$3



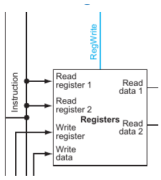
Obs. note que nessa versão do circuito não estamos armazenando o endereço de RS em ID/EX. Vamos precisar fazer isso!

se (ID/EX.RegistradorRS == EX/MEM.RegistradorRD) então Entrada1ALU = EX.MEM/DadoRD

Anotações

Hazards de Dados

- No exemplo a 1ª fonte da ALU deve vir de `EX/MEM`
  - O mesmo pode acontecer para a 2ª fonte da ALU
  - E o resultado pode vir ainda de `MEM/WB`
- Ainda temos o problema de que não é toda instrução que escreve nos registradores
  - O dado em `EX/MEM`, ou em `MEM/WB`, mesmo sendo mais recente, pode não fazer sentido
  - Como saber se o dado nesses estágios vai ser escrito no registrador?
    - Uma solução é verificar se o sinal de controle RegWrite está ativo para a instrução que se encontra no estágio EX ou MEM



Anotações

Forward do `$zero`

- Outro problema é se fizermos o forward do registrador `$zero`
- e.g. `addi $0, $1, 2`
- Podemos tratar esse problema de várias formas:
  - Especificar que o montador gera um erro nesse código Assembly:
    - Parece o correto, mas e se modificarmos o código de máquina diretamente? E quem garante que o montador vai fazer as coisas direito?
  - Podemos fazer com que o processador lance uma exceção
    - Algo que acontece também quando fazemos uma divisão por zero
    - Seria uma boa solução, mas vamos deixar exceções de lado por enquanto
  - Ou podemos efetivamente realizar o cálculo e "armazenar" em `$zero` (`$0`)
    - O que vai acontecer é que o banco de registradores vai ignorar esse resultado e manter `$0` com o valor `0`
    - **Mas devemos tomar cuidado**:

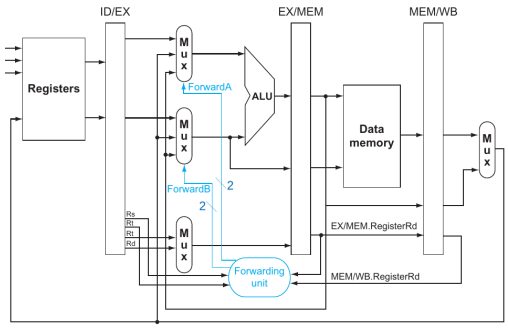
```
1 addi $0, $1, 2
2 sub $2, $3, $0
```

    - se fizermos os forward do `$0`, vamos usar um valor que será descartado!

Anotações

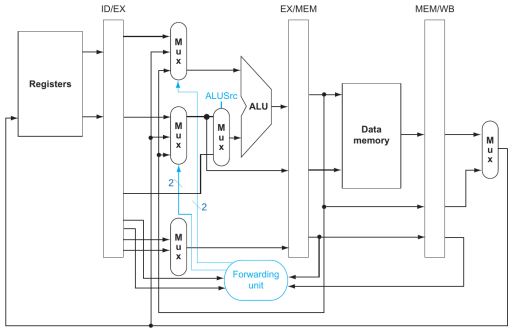
Unidade de Forwarding

- Para simplificar, essa versão do circuito não está com o multiplexador para escolher entre o registrador e o campo imediado como segundo operando da ALU. Como ficaria com este multiplexador?



Anotações

Unidade de Forwarding



Anotações

Unidade de Forwarding

Sinal MUX	Fonte	Descrição
ForwardA = 00	ID/EX	Primeiro operando da ALU deve vir do banco de registradores (sem *forward*)
ForwardA = 10	EX/MEM	Primeiro operando da ALU deve vir de EX/MEM (*forward*)
ForwardA = 01	MEM/WB	Primeiro operando da ALU deve vir de MEM/WB (*forward*)
ForwardB = 00	ID/EX	Segundo operando da ALU deve vir do banco de registradores (sem *forward*)
ForwardB = 10	EX/MEM	Segundo operando da ALU deve vir de EX/MEM (*forward*)
ForwardB = 01	MEM/WB	Segundo operando da ALU deve vir de MEM/WB (*forward*)

Anotações

Unidade de Forwarding

```
ForwardA = 00
ForwardB = 00
if EX/MEM.RegWrite and (EX/MEM.RegisterRd != 0) and (EX/MEM.RegisterRd = ID/EX.RegisterRs):
    ForwardA = 10
if EX/MEM.RegWrite and (EX/MEM.RegisterRd != 0) and (EX/MEM.RegisterRd = ID/EX.RegisterRt):
    ForwardB = 10
if MEM/WB.RegWrite and (MEM/WB.RegisterRd != 0) and (MEM/WB.RegisterRd = ID/EX.RegisterRs):
    ForwardA = 01
if MEM/WB.RegWrite and (MEM/WB.RegisterRd != 0) and (MEM/WB.RegisterRd = ID/EX.RegisterRt):
    ForwardB = 01
```

- `EX/MEM.RegWrite` e `MEM/WB.RegWrite`: Testa o sinal de controle para verificar se a instrução em `EX/MEM` e `MEM/WB` (respectivamente) pretende escrever o resultado no registrador
- `EX/MEM.RegisterRd != 0` e `MEM/WB.RegisterRd != 0`: O registrador de destino não pode ser o **`$zero`**
- `EX/MEM.RegisterRd = ID/EX.RegisterRs`, `EX/MEM.RegisterRd = ID/EX.RegisterRt`, `MEM/WB.RegisterRd = ID/EX.RegisterRs` e `MEM/WB.RegisterRd = ID/EX.RegisterRt`: Testa se o endereço do registrador da 1ª e 2ª fonte no estágio `EX` é o mesmo de destino das instruções nos estágios `MEM` e/ou `WB`.

Anotações

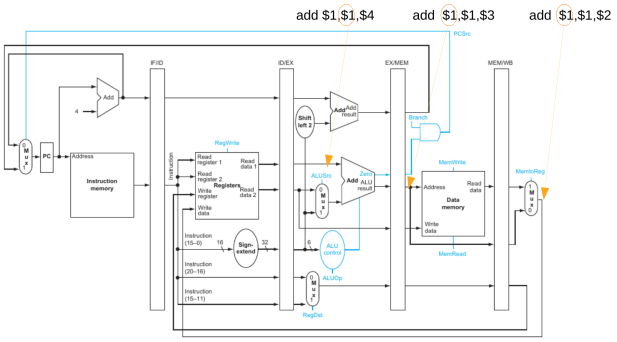
Mais Complicações

- O resultado ainda não salvo de um registrador pode esta em `EX/MEM`, e também em `MEM/WB`

```
1 add $1,$1,$2
2 add $1,$1,$3
3 add $1,$1,$4
```

Anotações

Mais Complicações

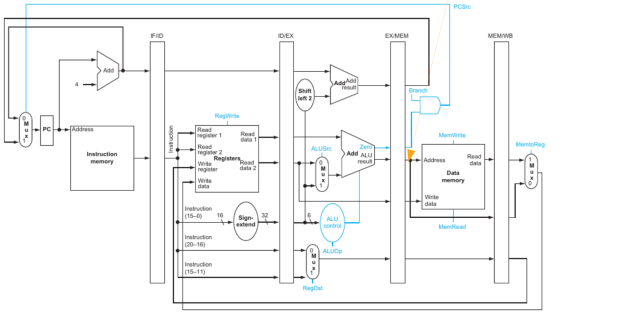


Anotações

Mais Complicações

Esse é o resultado mais recente!

add $1,$1,$4 add $1,$1,$3 add $1,$1,$2



Anotações

Condições atualizadas

```
ForwardA = 00
ForwardB = 00

if EX/MEM.RegWrite and (EX/MEM.RegisterRd != 0) and (EX/MEM.RegisterRd = ID/EX.RegisterRs):
    ForwardA = 10
if EX/MEM.RegWrite and (EX/MEM.RegisterRd != 0) and (EX/MEM.RegisterRd = ID/EX.RegisterRt):
    ForwardB = 10
if MEM/WB.RegWrite and (MEM/WB.RegisterRd != 0) and (MEM/WB.RegisterRd = ID/EX.RegisterRs)
   and not(EX/MEM.RegWrite and (EX/MEM.RegisterRd != 0) and (EX/MEM.RegisterRd != ID/EX.RegisterRs)):
    ForwardA = 01
if MEM/WB.RegWrite and (MEM/WB.RegisterRd != 0) and (MEM/WB.RegisterRd = ID/EX.RegisterRt)
   and not(EX/MEM.RegWrite and (EX/MEM.RegisterRd != 0) and (EX/MEM.RegisterRd != ID/EX.RegisterRt)):
    ForwardB = 01
```

- Dar preferência ao resultado no estágio `MEM` (registrador de pipeline `EX/MEM`)
  - Mais recente

Anotações

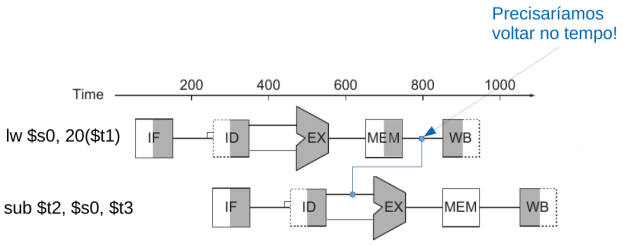
Unidade de forwarding

- A unidade desenvolvida serve para (os dados necessários no)/o estágio `EX`
- Hazards de dados ainda podem acontecer para (os dados necessários no)/o estágio `MEM`
  - e.g., um **`lw`** seguido de um **`sw`** se referenciando ao mesmo endereço de memória
- A unidade de forwarding em `MEM` é mais simples do que a em `EX`

Anotações

Stalls causados por hazards de dados

- Não é possível solucionar qualquer hazard de dados através de forwardings



Anotações

Stalls causados por hazards de dados

- Não é possível solucionar qualquer hazard de dados através de forwardings
- Nesses casos precisamos de um pipeline stall
  - Inserir bolhas no pipeline
  - Uma bolha é inserida efetivamente inserindo-se uma instrução que não executa operação alguma
  - Esse tipo de instrução geralmente é chamada de `nop` (no operation)
    - Não escreve em nenhum registrador (Incluindo o PC, que não é atualizado)
    - Não escreve na memória
    - Não altera as informações nos registradores de pipeline

Anotações

Stalls causados por hazards de dados

- Nosso processador e pipeline são simples
  - A única combinação que causará stalls são loads seguidos de alguma instrução que usa o conteúdo do registrador sendo carregado

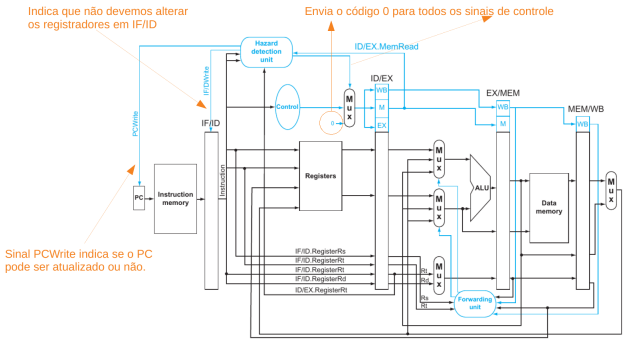
```
1 lw $s1, 4($s2)
2 add $s4, $s1, $s5
```

- E o stall é de somente uma instrução
  - Não precisamos adicionar mais de um `nop` no meio das instruções
- O controle da unidade de detecção pode ser o seguinte

```
if ID/EX.MemRead and (
        (ID/EX.RegisterRt = IF/ID.RegisterRs) or
        (ID/EX.RegisterRt = IF/ID.RegisterRt)
    ):
    pipeline stall
```

Anotações

Stalls causados por hazards de dados



Anotações

Exercícios

1. Volte nas aulas anteriores e verifique que ao enviar zero em todos os sinais de controle, nada será alterado ao final de uma instrução
2. Usando os slides como exemplo, crie uma unidade de forwaring para o estágio MEM do pipeline.

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.

Anotações