"Como assim por que teve de ser criado? É um bypass (atalho). Você precisa criar bypasses." (Douglas Adams; O Guia do Mochileiro das Galáxias, 1979).

Construindo Forwardings (Bypasses)

Paulo Ricardo Lisboa de Almeida

DSBD

UFPR
UNIVERSIDADE FEDERAL DO PARANÁ

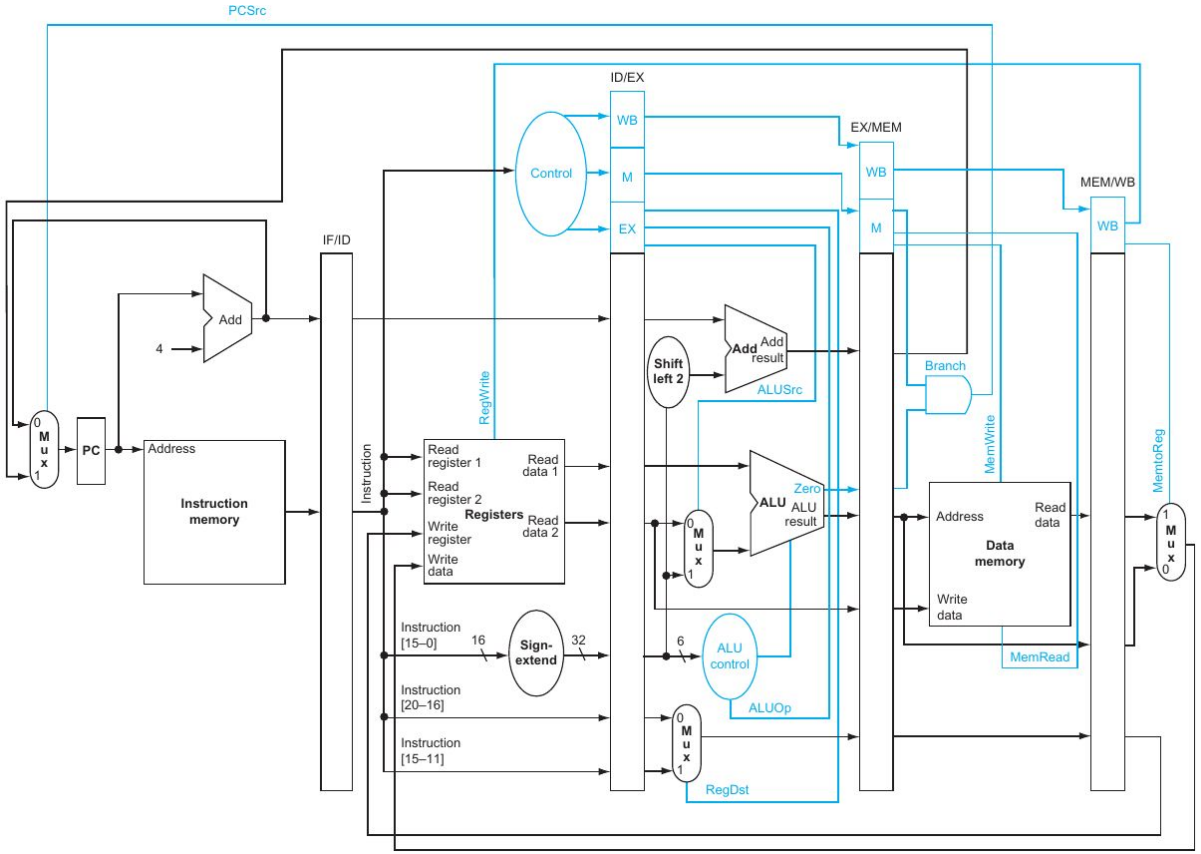
Hazards de Dados

O pipeline construído até o momento funciona caso as instruções não tenham dependências.

Mas **hazards de dados e de controle são comuns.**

Hazards estruturais já foram resolvidos.

Começando pelo tratamento dos hazards de dados via **Forwardings.**



Exemplo

Considere as instruções a seguir

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

Onde estão os hazards de dados?

Exemplo

Considere as instruções a seguir

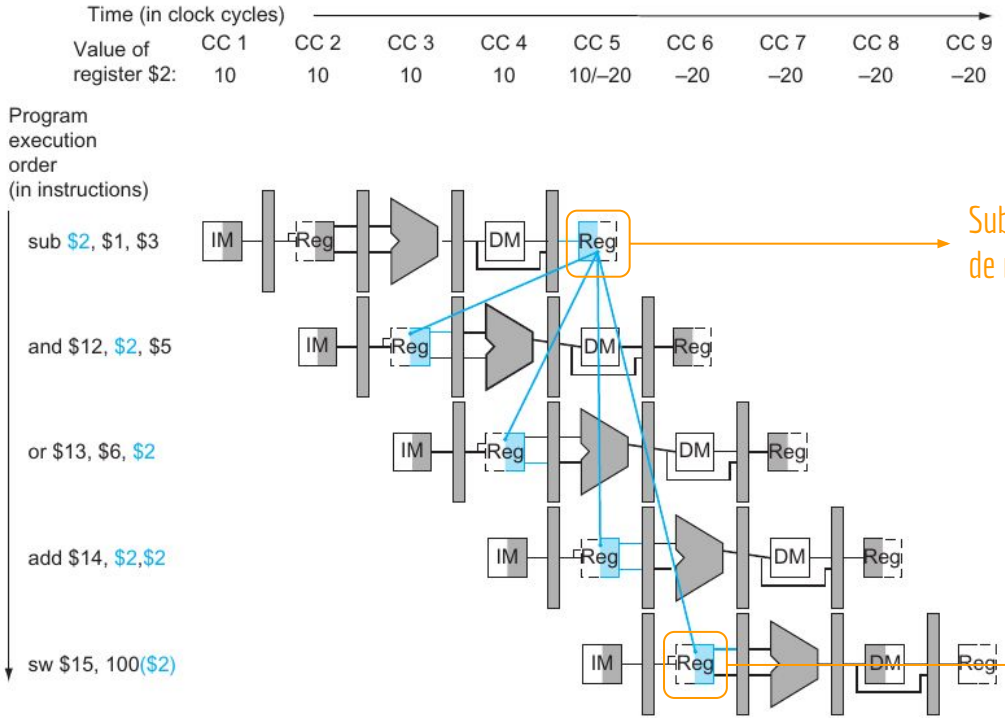
```
sub $2,$1,$3
and $12,$2,$5
or $13,$6,$2
add $14,$2,$2
sw $15,100($2)
```

Onde estão os hazards de dados?

As instruções marcadas dependem do resultado do sub.

Se isso vai causar hazards de dados ou não depende diretamente de como o pipeline é montado, e de sua profundidade.

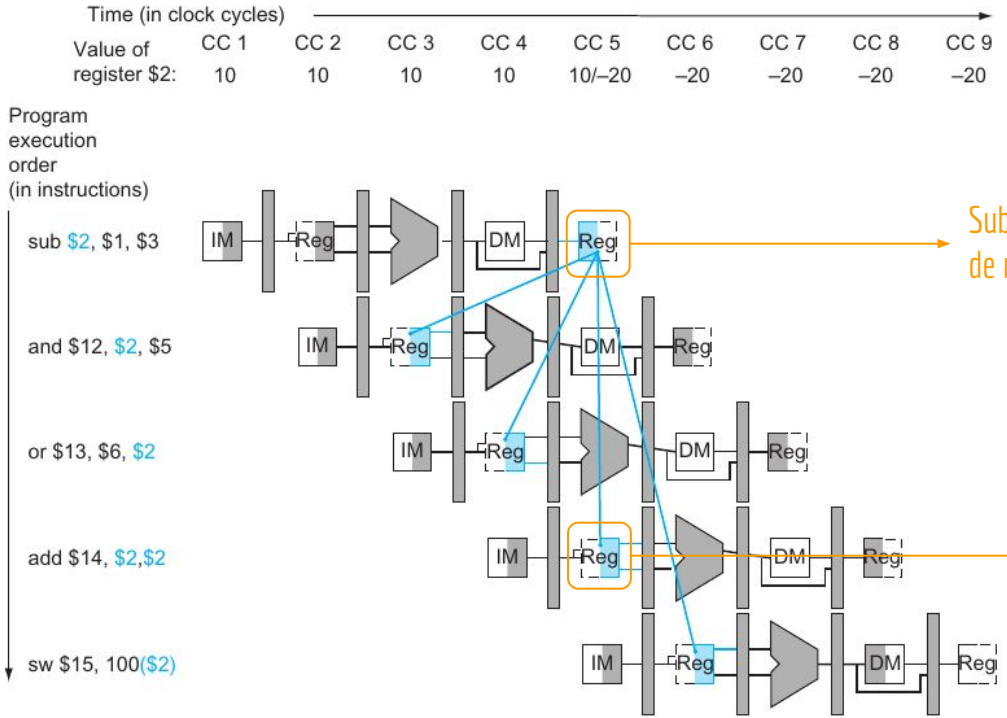
Considerando o Pipeline do MIPS32



Sub armazena o resultado de $2 no banco de registradores no ciclo de clock 5.

O SW está lê $2 no ciclo 6, então não há hazard.

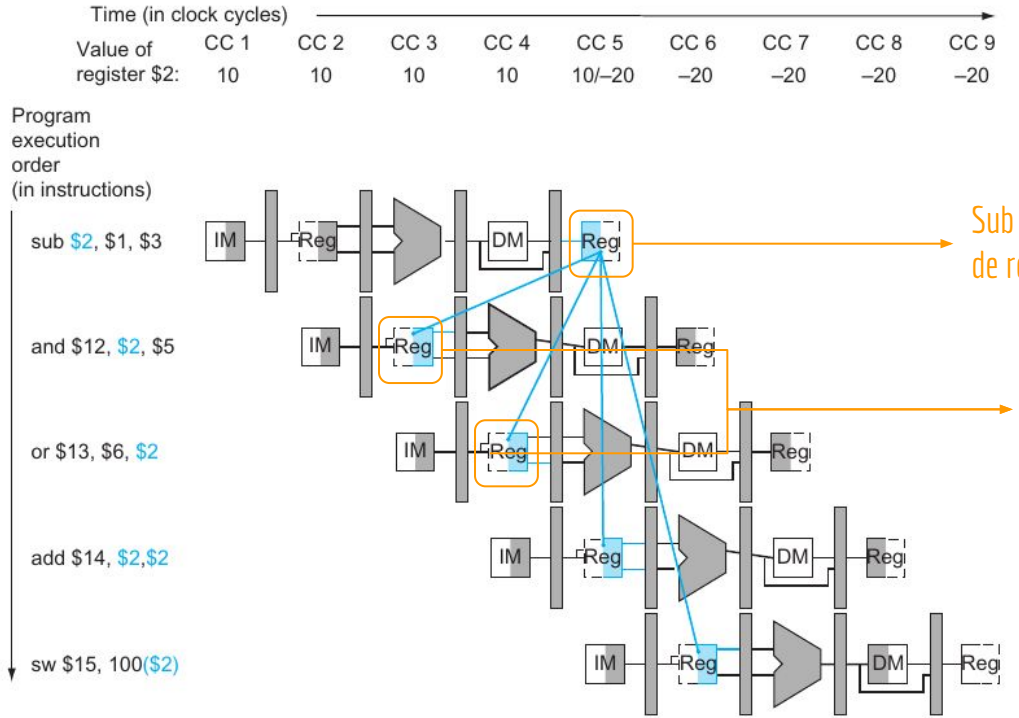
Considerando o Pipeline do MIPS32



Sub armazena o resultado de $2 no banco de registradores no ciclo de clock 5.

O add está lendo $2 no mesmo ciclo em que o resultado está sendo escrito. Mas isso não gera um hazard no banco de registradores. Os flip-flops podem ser construídos de forma que o dado é escrito no início do ciclo, e lidos no restante do ciclo.

Considerando o Pipeline do MIPS32



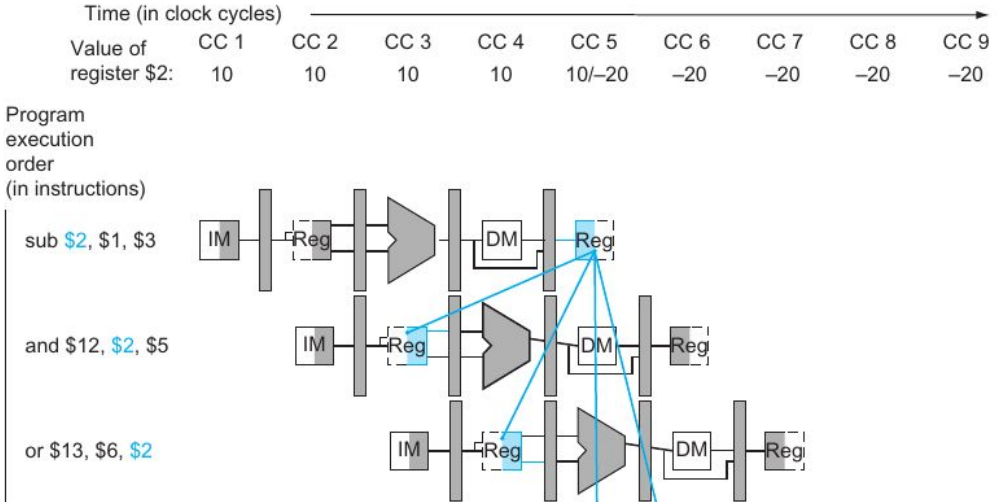
Sub armazena o resultado de $2 no banco de registradores no ciclo de clock 5.

Hazards de dados. O and e o or tentam usar $2 antes do resultado estar pronto.

Considerando o Pipeline do MIPS32

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)?



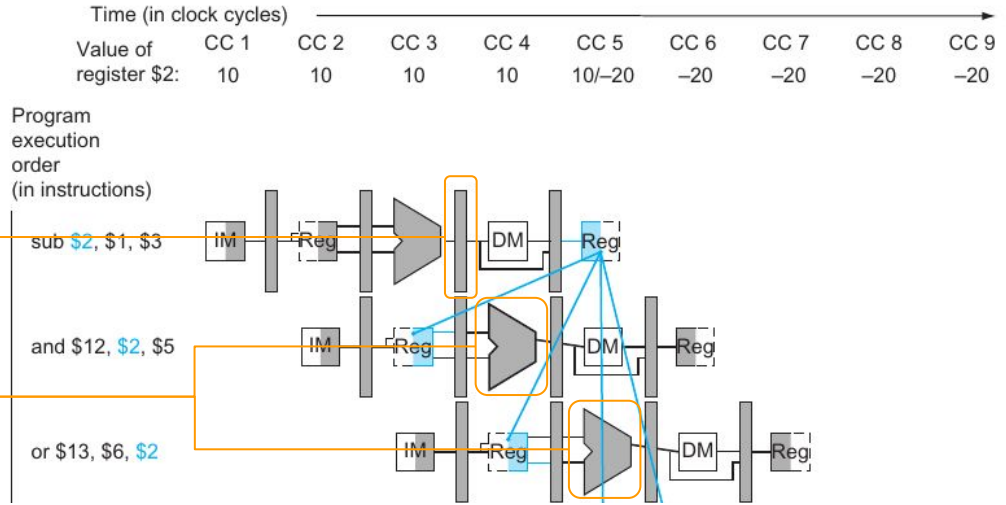
Considerando o Pipeline do MIPS32

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)?

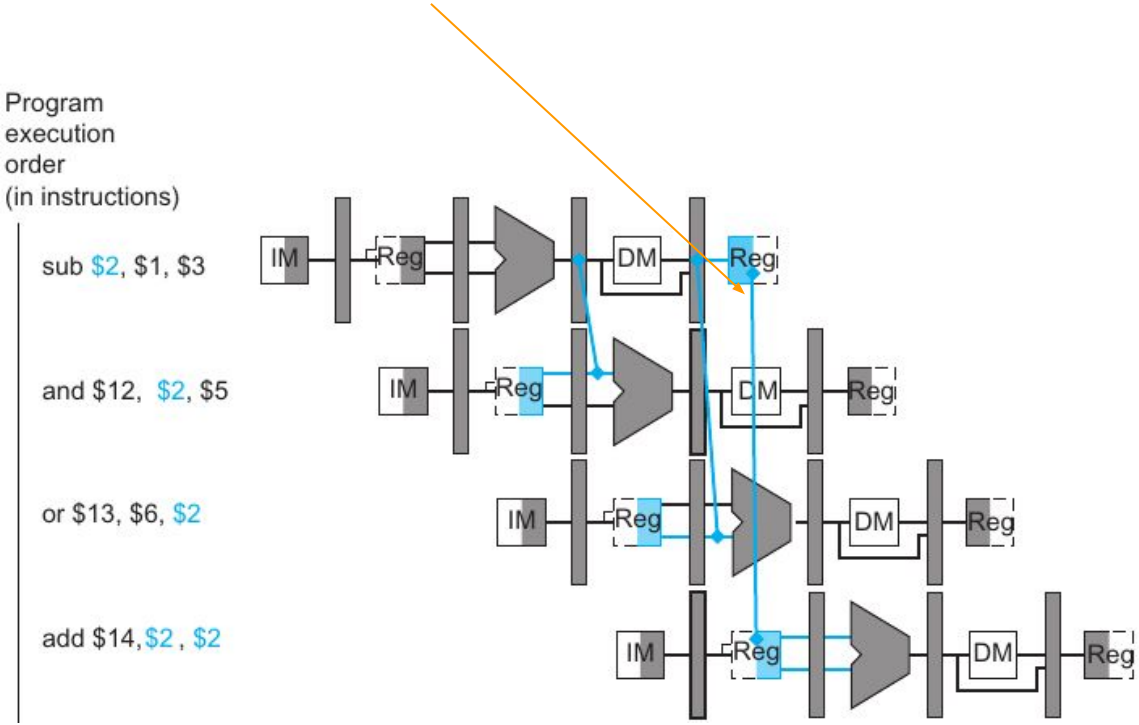
O resultado está pronto quando sai da ALU.

O valor é utilizado na ALU nas demais instruções.



Considerando o Pipeline do MIPS32

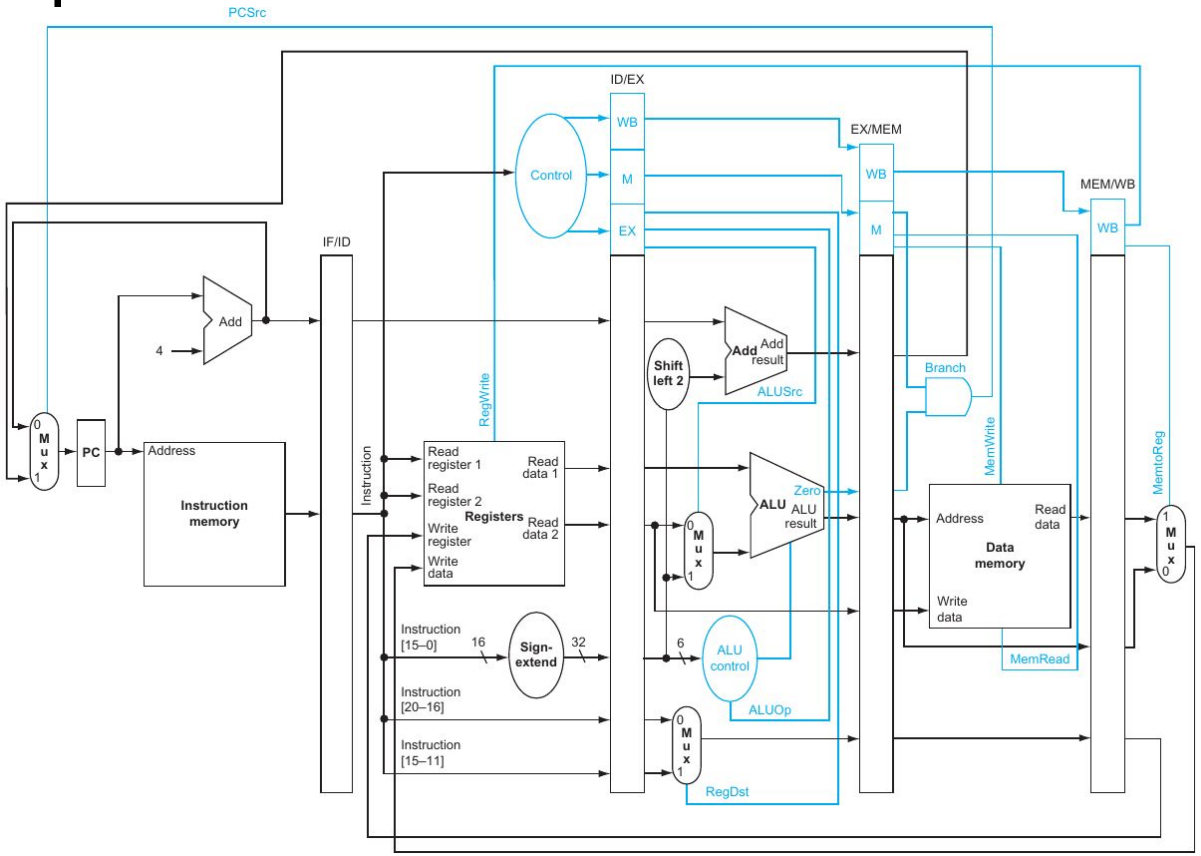
Forward feito "automaticamente" pelo banco de registradores.



Registradores de Pipeline

Os registradores de pipeline armazenam as informações relevantes da instrução a cada estágio.

Nomes dos registradores de pipeline:
IF/ID,ID/EX, EX/MEM, MEM/WB.

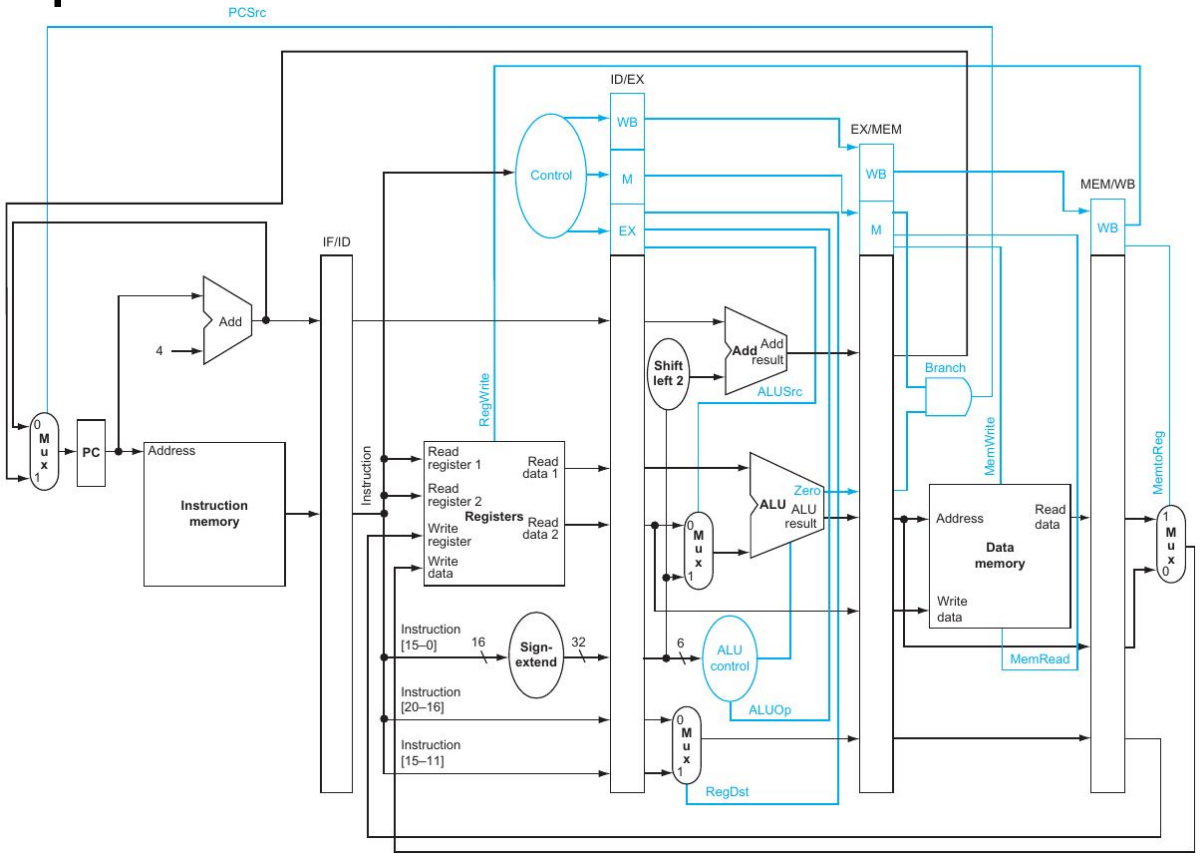


Registradores de Pipeline

Os registradores de pipeline armazenam as informações relevantes da instrução a cada estágio.

Nomes dos registradores de pipeline:
IF/ID,ID/EX, EX/MEM, MEM/WB.

Para referenciar a informação armazenada nos registradores de pipeline, será utilizada a seguinte notação: *nomeRegPipeline.nomeDado*



Registradores de Pipeline

Os registradores de pipeline armazenam as informações relevantes da instrução a cada estágio.

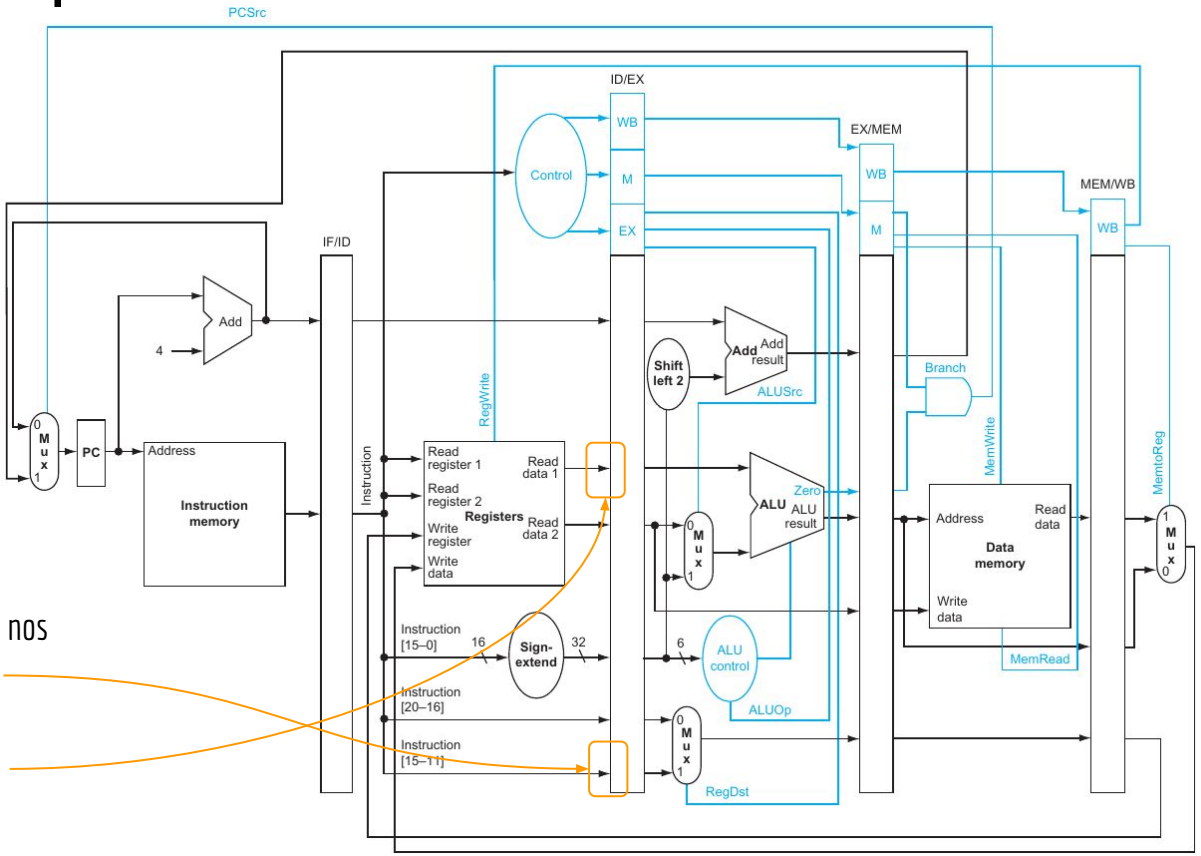
Nomes dos registradores de pipeline:
IF/ID,ID/EX, EX/MEM, MEM/WB.

Para referenciar a informação armazenada nos registradores de pipeline, será utilizada a seguinte notação: *nomeRegPipeline.nomeDado*

Exemplos:

Endereço do registrador destino, armazenado nos registradores ID/EX: **ID/EX.RegistradorRD**

Dado do registrador fonte 1, armazenado nos registradores ID/EX: **ID/EX.DadoRS**

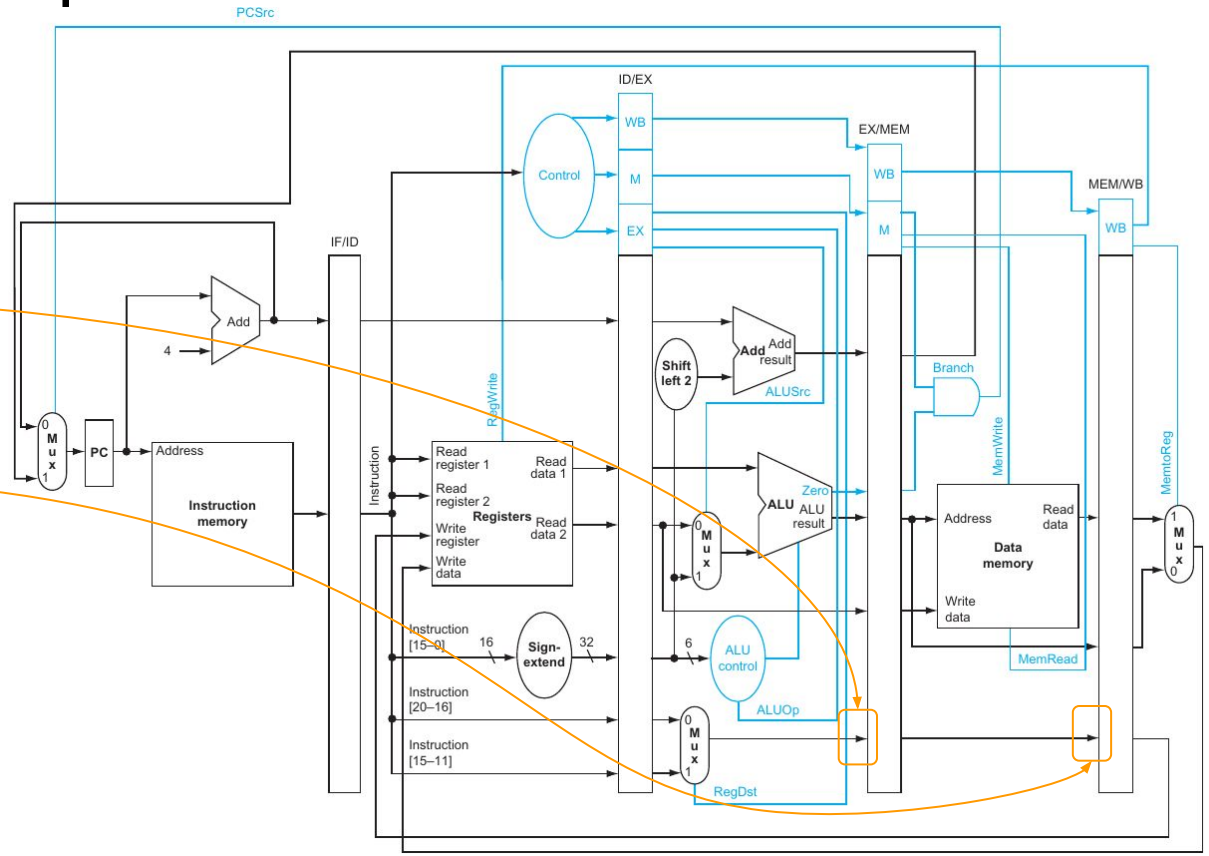


Registradores de Pipeline

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 como: MEM/WB.RegistradorRD

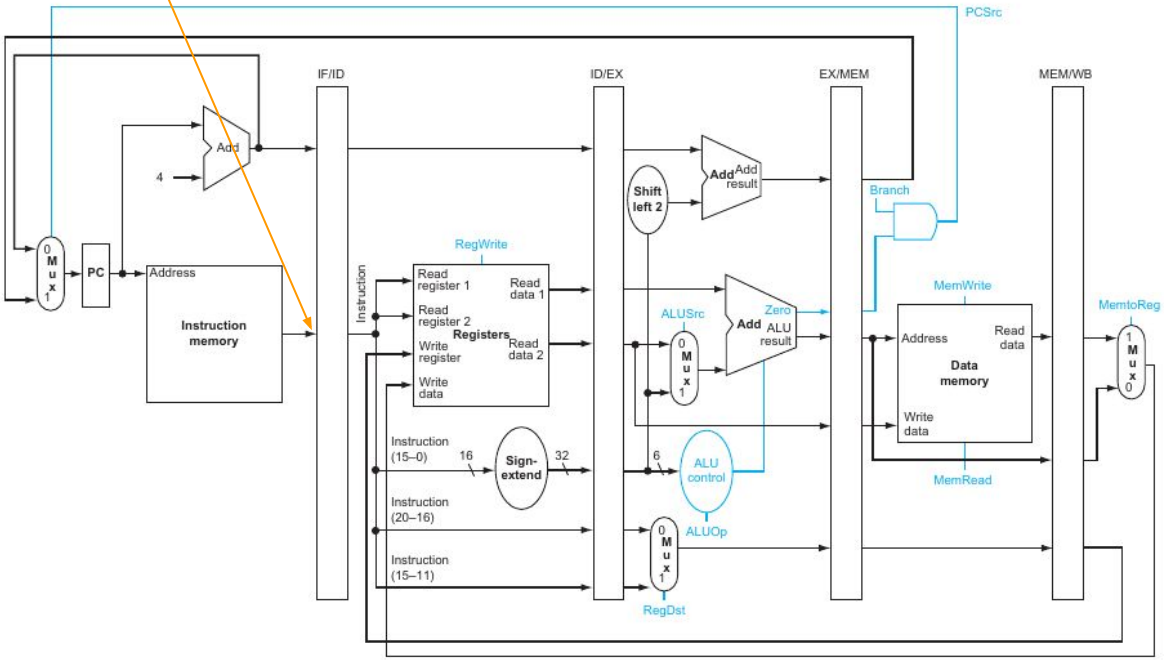


Exemplo passo a passo

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.



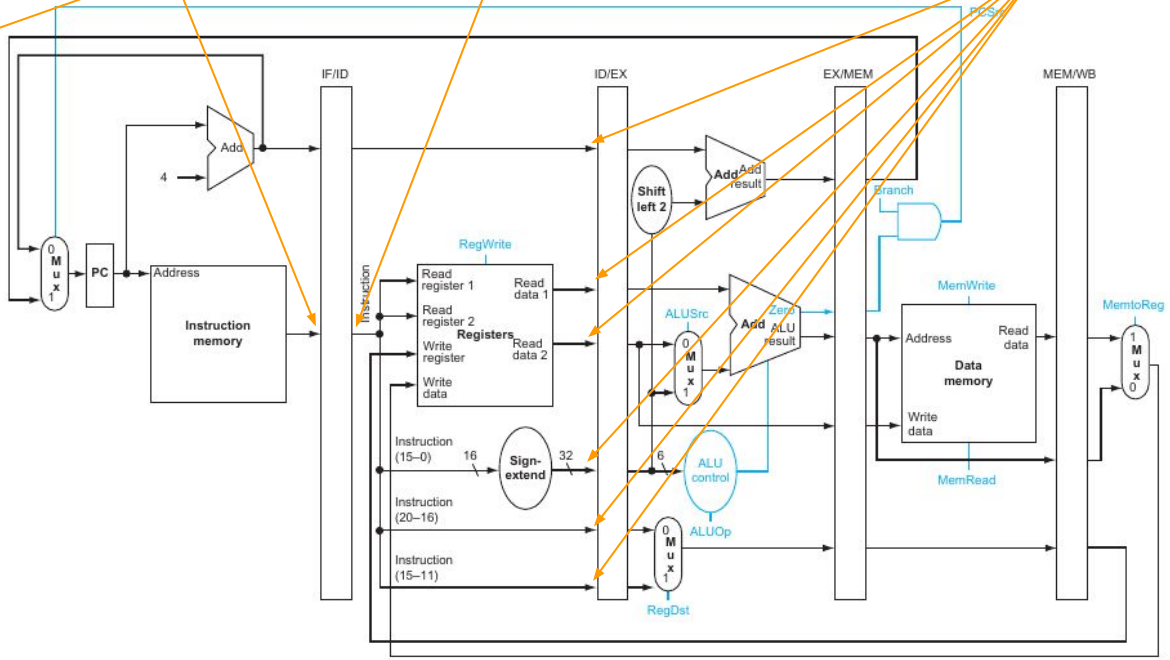
Exemplo passo a passo

A instrução salva anteriormente é carregada para continuar.

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

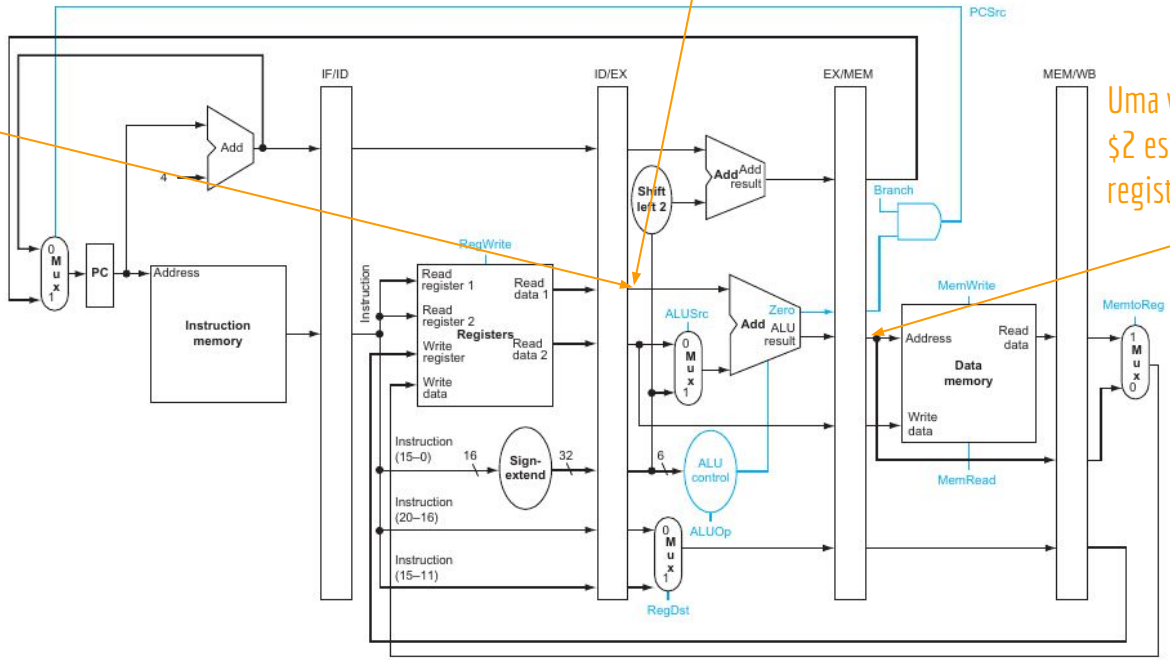
Os dados referentes ao sub são armazenados agora em ID/EX.

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



Exemplo passo a passo

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



O and está pegando $2 que foi salvo anteriormente.

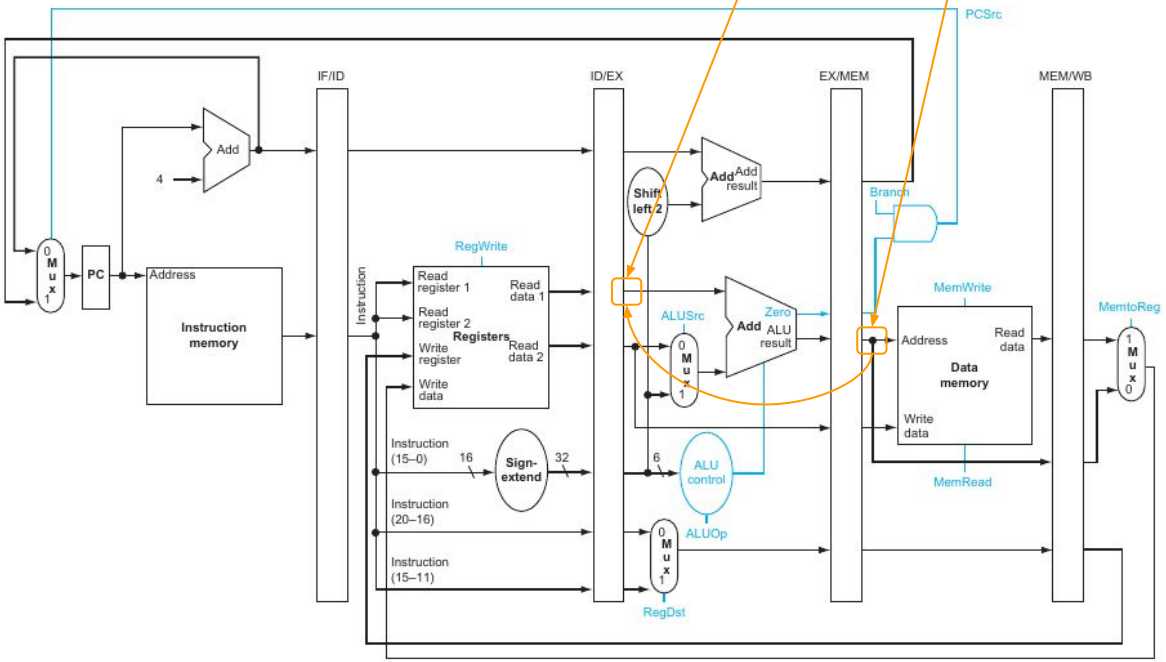
Uma versão mais atual de $2 está presente nos registradores EX/MEM.

Aqui as coisas dão errado!

Forwarding

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

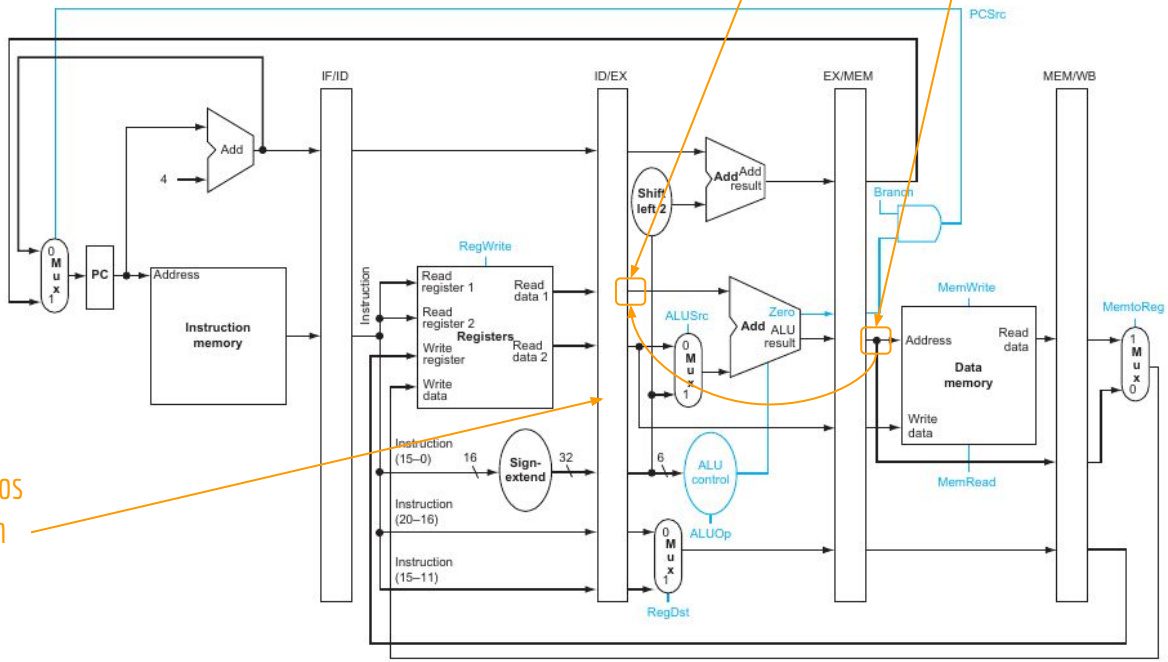
... ... and $12, $2, $5 sub $2, $1, $3



Forwarding

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

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



Nessa versão do circuito não estamos armazenando o **endereço** de RS em ID/EX. Vamos precisar fazer isso!

Hazard de Dados

No exemplo, o 1° fonte da ALU deve vir de EX/MEM.

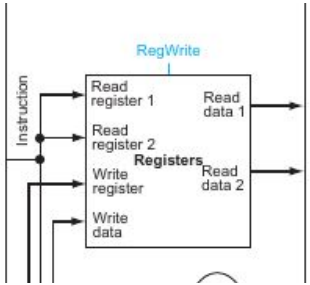
O mesmo pode acontecer para o 2 fonte.

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?



Hazard de Dados

No exemplo, o 1° fonte da ALU deve vir de EX/MEM.

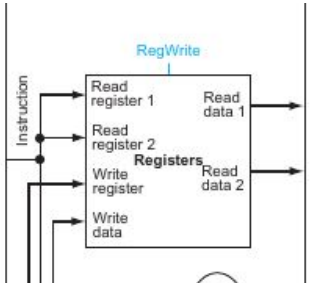
O mesmo pode acontecer para o 2 fonte.

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**.

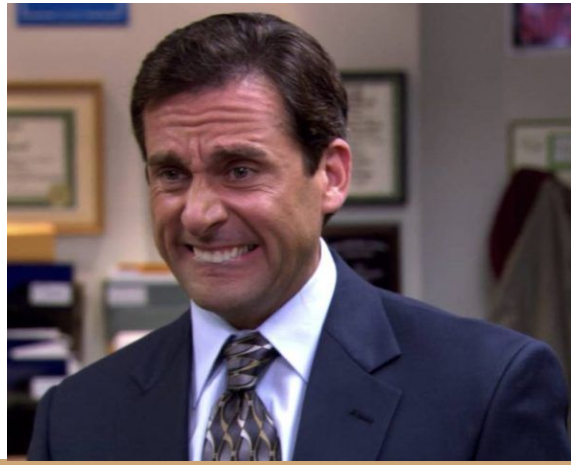
Uma solução é verificar se o sinal de controle RegWrite está ativo para a instrução que se encontra no estágio EX ou MEM.



Forward do $zero

Outro problema é se fizermos o forward do registrador $zero.

Exemplo: `addi $0,$1,2`



Forward do $zero

Outro problema é se fizermos o forward do registrador $zero.

Exemplo: `addi $0,$1,2`

Podemos tratar esse problema de várias formas.

Especificar que o **montador** gera um erro nesse código Assembly.

Problemas?

Forward do $zero

Outro problema é se fizermos o forward do registrador $zero.

Exemplo: `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?

Quem garante que o montador vai fazer as coisas direito?

Parece que estamos delegando o problema para quem não deveria ser o responsável.

Forward do $zero

Outro problema é se fizermos o forward do registrador $zero.

Exemplo: `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?

Quem garante que o montador vai fazer as coisas direito?

Parece que estamos delegando o problema para quem não deveria ser o responsável.

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 para o futuro.

Forward do $zero

Ou podemos efetivamente realizar o cálculo e “armazenar” em $zero ($0).

O banco de registradores vai ignorar esse resultado e manter $0 com o valor 0.

Forward do $zero

Ou podemos efetivamente realizar o cálculo e “armazenar” em $zero ($0).

O banco de registradores vai ignorar esse resultado e manter $0 com o valor 0.

Onde está o problema com os forwardings nesse caso?

Forward do $zero

Ou podemos efetivamente realizar o cálculo e “armazenar” em $zero ($0).

O banco de registradores vai ignorar esse resultado e manter $0 com o valor 0.

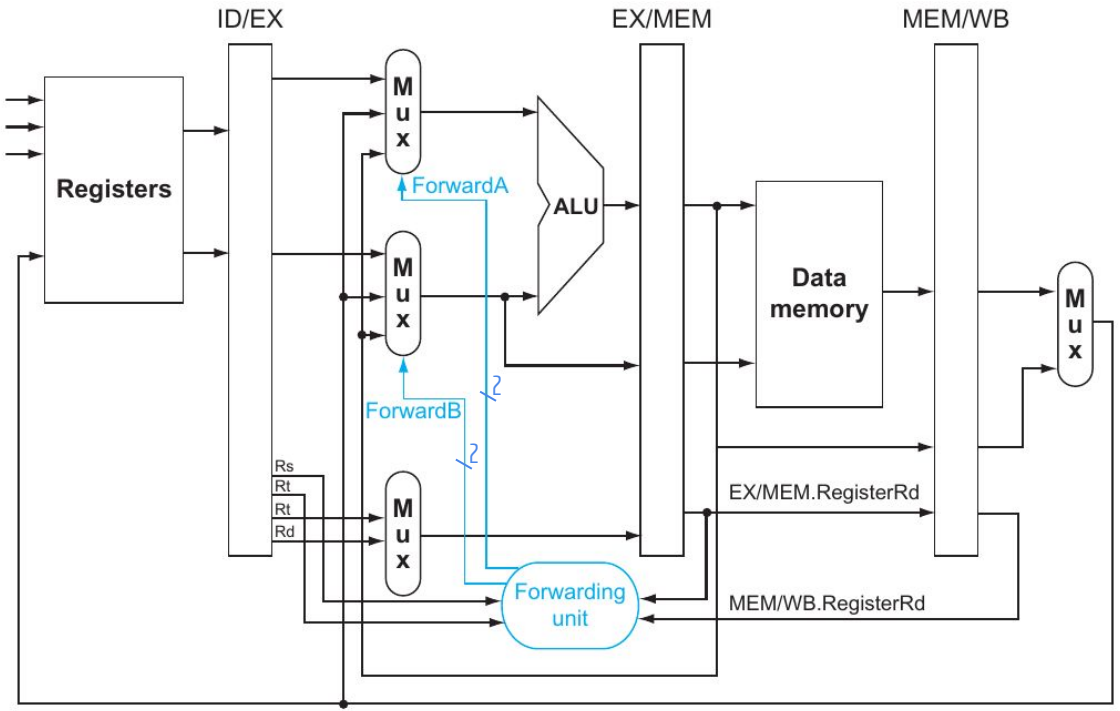
Onde está o problema com os forwardings nesse caso?

```
addi $0,$1,2
sub $2,$3,$0 #se fizermos os forward do $0, será usado um valor que será descartado!
```

Unidade de Forwarding

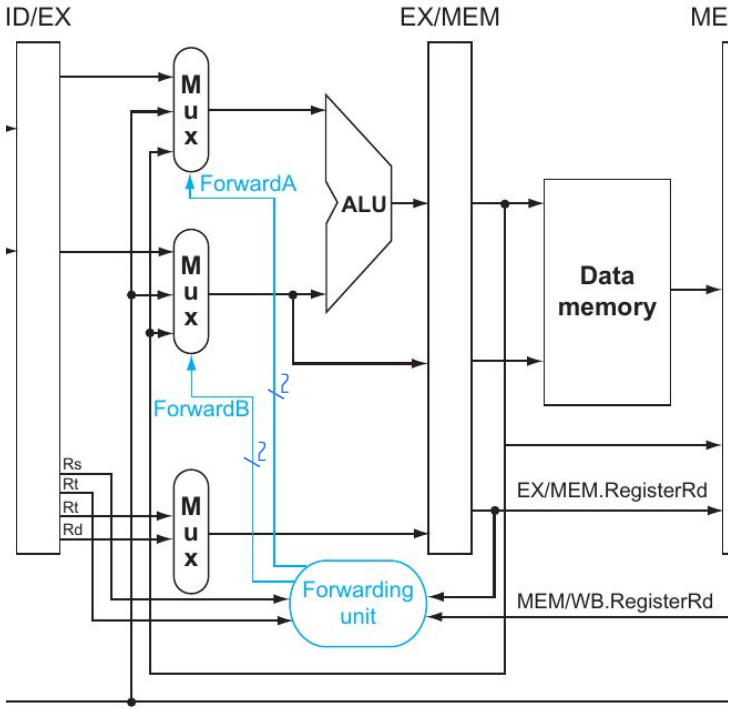
Para simplificar, essa versão do circuito não está com o multiplexador para escolher entre o registrador e o campo imediato como segundo operando da ALU.

Faça você mesmo: Adicione esse multiplexador e compare com a solução do livro.



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	Primeiro operando da ALU deve vir do banco de registradores (sem forward).
ForwardB = 10	EX/MEM	Primeiro operando da ALU deve vir de EX/MEM (forward).
ForwardB = 01	MEM/WB	Primeiro operando da ALU deve vir de MEM/WB (forward).



Condições

As condições testadas pela unidade de forwarding são algo como:

Verifica se a instrução em EX/MEM vai escrever o resultado no registrador.

O registrador de destino não pode ser o $zero.

Testa se o endereço do 1° fonte é o mesmo do registrador destino da instrução no estágio EX.

```
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
```

Mais Problemas

O resultado ainda não salvo de um registrador pode estar em EX/MEM, e **também** em MEM/WB.

Exemplo:

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

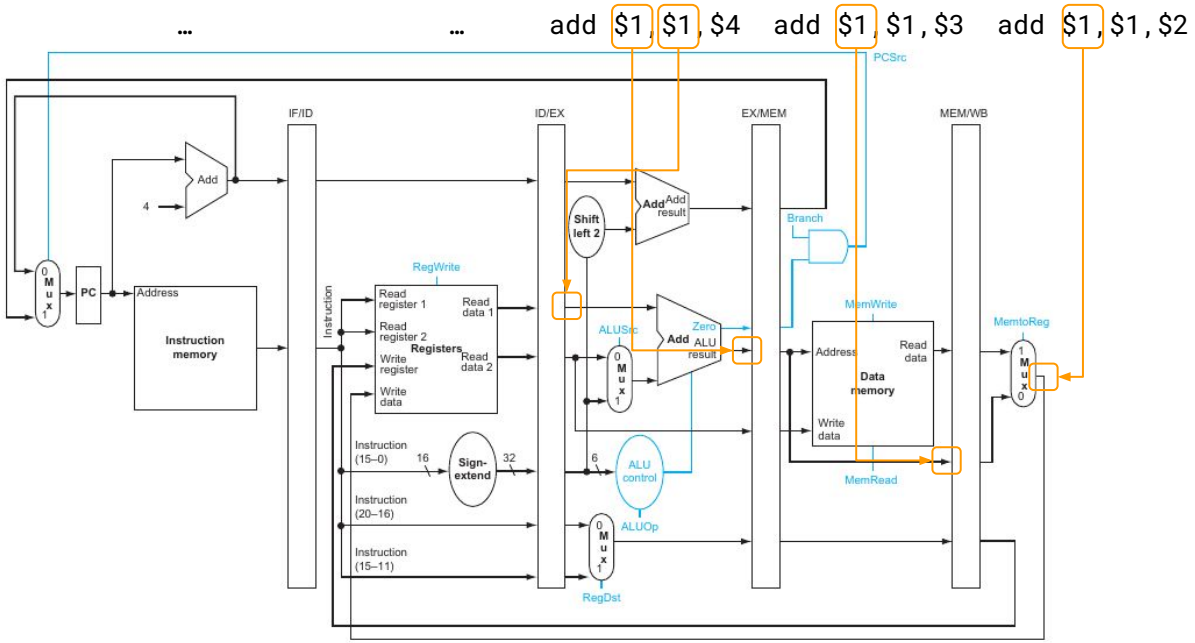
Mais Problemas

O resultado ainda não salvo de um registrador pode estar em EX/MEM, e **também** em MEM/WB.

Exemplo:

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

Qual versão usar?

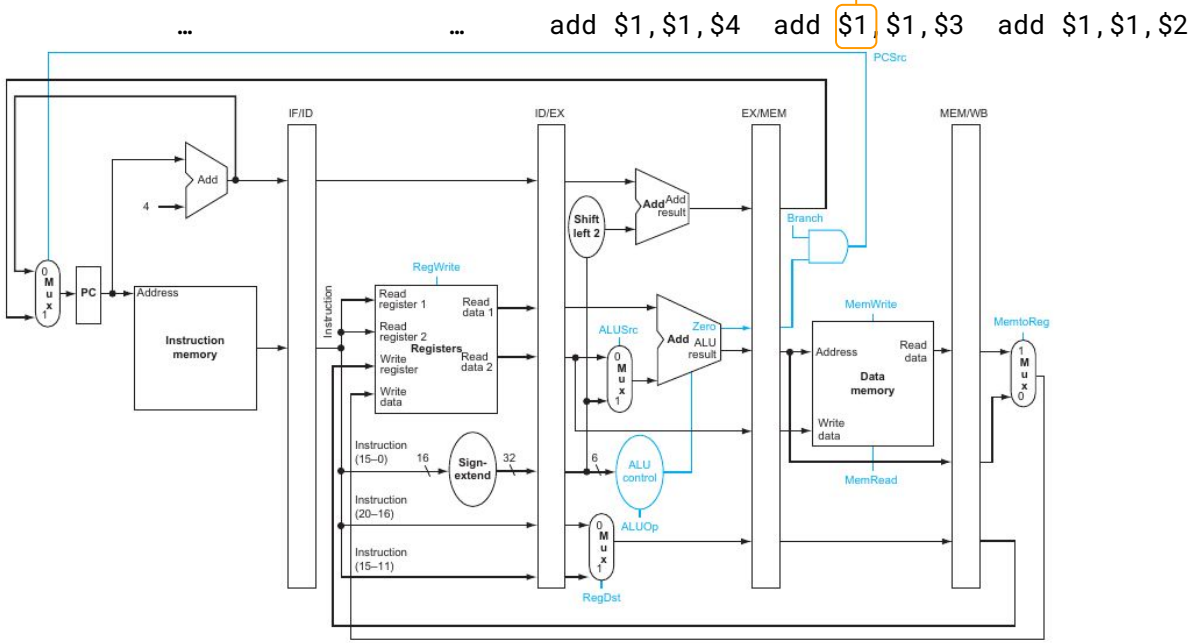


Mais Problemas

O resultado ainda não salvo de um registrador pode estar em EX/MEM, e **também** em MEM/WB.

Exemplo:

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



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 not(EX/MEM.RegWrite 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 and (MEM/WB.RegisterRd ≠ 0)
      and not(EX/MEM.RegWrite and (EX/MEM.RegisterRd ≠ 0) and (EX/MEM.RegisterRd ≠ ID/EX.RegisterRt))
      and (MEM/WB.RegisterRd = ID/EX.RegisterRt))
      ForwardB = 01
```

Unidade de forwarding

A unidade desenvolvida serve para o estágio EX.

Hazards de dados ainda podem acontecer no estágio MEM.

Exemplo: um *lw* seguido de um *sw* se referenciando ao mesmo endereço de memória.

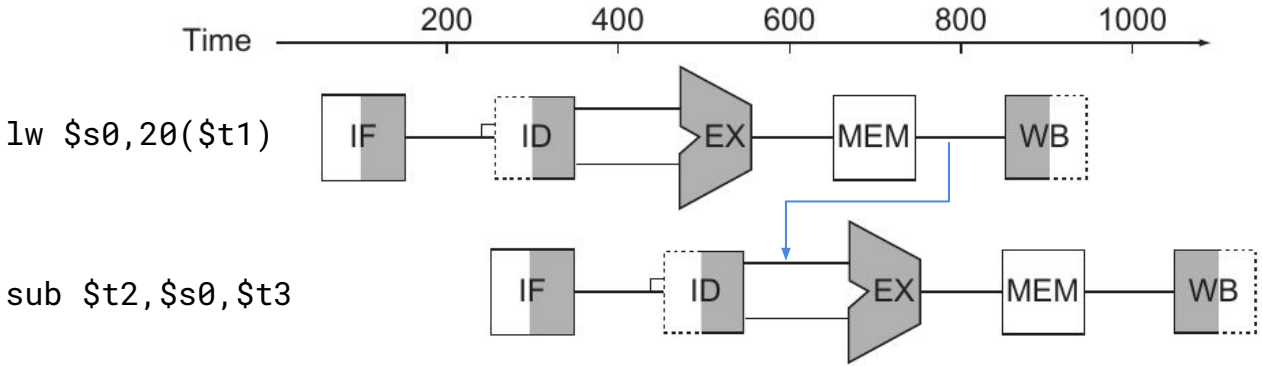
A unidade de forwarding em MEM é mais simples do que a em EX.

Exercício: defina essa unidade você mesmo.

Stalls causados por hazards de dados

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

Exemplo:



Stalls causados por hazards de dados

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

Precisamos de um **pipeline stall**.

- Inserir bolhas no pipeline.
- Uma bolha é inserida efetivamente com uma **instrução que não executa operação alguma**.
  - Esse tipo de instrução geralmente é chamada de **nop** – no operation.
    - Não escreve em nenhum registrador.
    - PC, que não é atualizado.
    - Não escreve na memória.
    - Não altera as informações nos registradores de pipeline.

Stalls causados por hazards de dados

Nosso processador e pipeline são simples.

A única combinação que causa stalls são loads seguidos de alguma instrução que usa o conteúdo do registrador carregado.

Exemplo:

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

O stall é de uma instrução.

Não precisamos adicionar mais de um *nop* no meio das instruções.

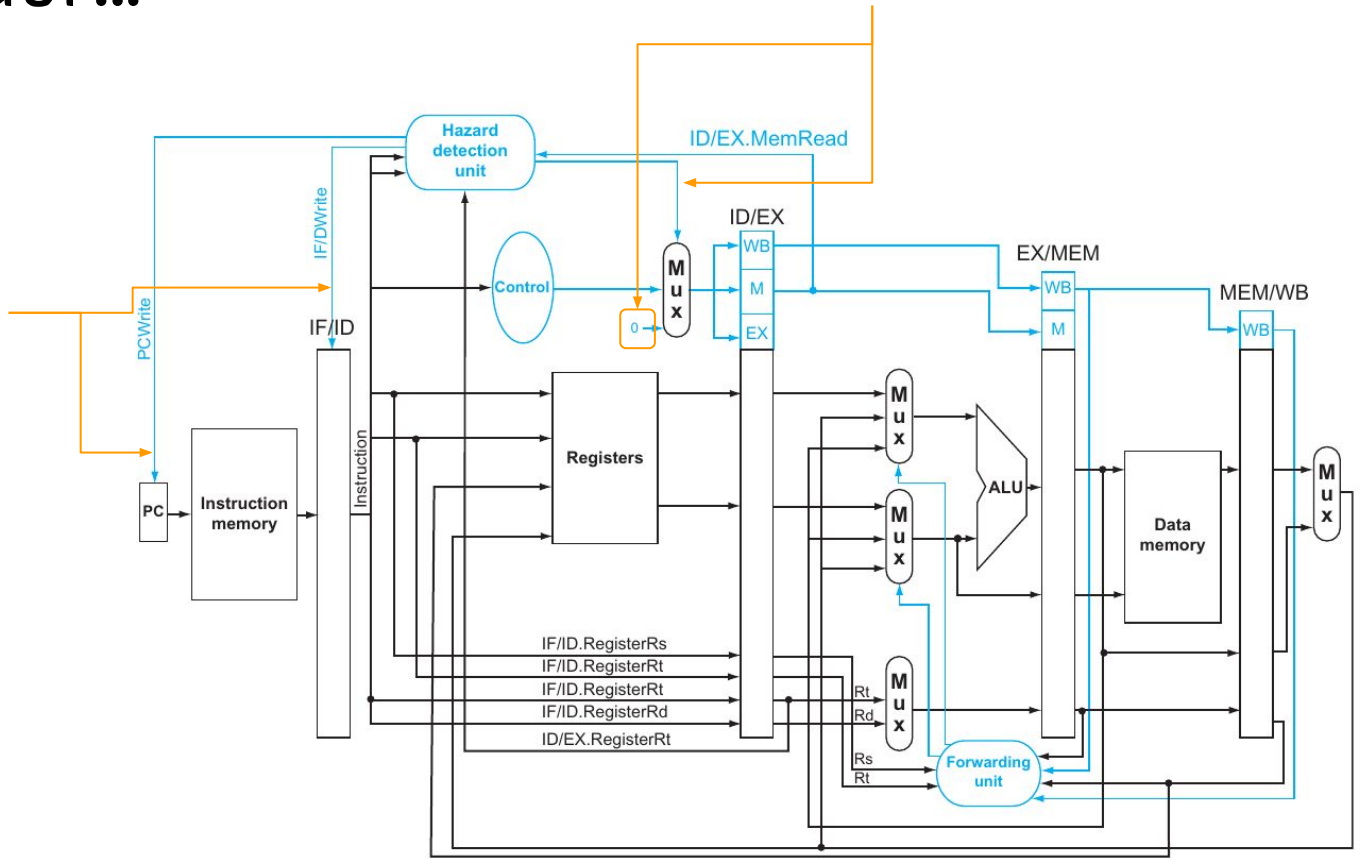
O controle da unidade de detecção de hazards pode ser o seguinte:

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

No processador...

Envia o código 0 para todos os sinais de controle.

Bits de enable para desabilitar a escrita.



Pipeline não é fácil!

Criar o pipeline e tratar os seus hazards não é fácil, mesmo em uma CPU simples.

Imagina tratar disso em um Pentium 4 Prescott!

Microprocessador	Ano	Clock	Estágios Pipeline	Tamanho Despacho	Fora de ordem?	CPUs por Chip	Potência
486	1989	0,025 GHz	5	1	Não	1	5 W
Pentium	1993	0,066 GHz	5	2	Não	1	10 W
Pentium Pro	1997	0,2 GHz	10	3	Sim	1	29 W
Pentium 4 Willamette	2001	2 GHz	22	3	Sim	1	75 W
Pentium 4 Prescott	2004	3,6 GHz	31	3	Sim	1	103 W
Intel Core	2006	3 GHz	14	4	Sim	2	75 W
Core i7 Nehalem	2008	3,6 GHz	14	4	Sim	2-4	87 W
Core Westmere	2010	3,73 GHz	14	4	Sim	6	130 W
Core i7 Ivy Bridge	2012	3,4 GHz	14	4	Sim	6	130 W
Core Broadwell	2014	3,7 GHz	14	4	Sim	10	140 W
Core i9 Skylake	2016	3,1 GHz	14	4	Sim	14	165 W
Intel Ice Lake	2018	4,2 GHz	14	4	Sim	16	185 W

Pipeline não é fácil!

Criar o pipeline e tratar os seus hazards não é fácil, mesmo em uma CPU simples.

Imagina tratar disso em um Pentium 4 Prescott!

Obs.: é o pipeline do Pentium 4 que vai cair no exame.

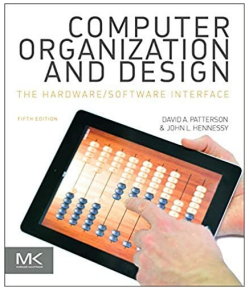
Microprocessador	Ano	Clock	Estágios Pipeline	Tamanho Despacho	Fora de ordem?	CPUs por Chip	Potência
486	1989	0,025 GHz	5	1	Não	1	5 W
Pentium	1993	0,066 GHz	5	2	Não	1	10 W
Pentium Pro	1997	0,2 GHz	10	3	Sim	1	29 W
Pentium 4 Willamette	2001	2 GHz	22	3	Sim	1	75 W
Pentium 4 Prescott	2004	3,6 GHz	31	3	Sim	1	103 W
Intel Core	2006	3 GHz	14	4	Sim	2	75 W
Core i7 Nehalem	2008	3,6 GHz	14	4	Sim	2-4	87 W
Core Westmere	2010	3,73 GHz	14	4	Sim	6	130 W
Core i7 Ivy Bridge	2012	3,4 GHz	14	4	Sim	6	130 W
Core Broadwell	2014	3,7 GHz	14	4	Sim	10	140 W
Core i9 Skylake	2016	3,1 GHz	14	4	Sim	14	165 W
Intel Ice Lake	2018	4,2 GHz	14	4	Sim	16	185 W

Exercícios

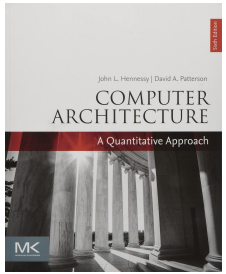
1. Releia os slides, discutindo cada ponto de forma detalhada com os colegas.
2. Volte nas aulas anteriores e verifique que ao enviar zero em todos os sinais de controle, nada será alterado ao final de uma instrução.
3. Usando os slides como exemplo, crie uma unidade de forwarding para o estágio MEM do pipeline.

Referências

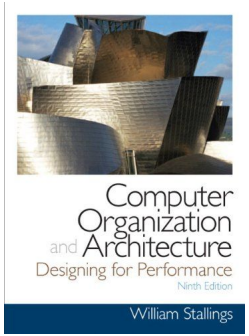
Patterson, Hennessy . Arquitetura e Organização de Computadores: A interface hardware/software. 2014.



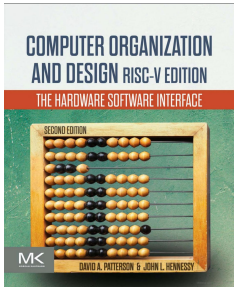
Hennessy, Patterson. Arquitetura de Computadores: uma abordagem quantitativa. 2019.



Stallings, W. Organização de Arquitetura de Computadores. 10a Ed. 2016.



Patterson, Hennessy. Computer Organization and Design RISC-V Edition: The Hardware Software Interface. 2020.



Licença