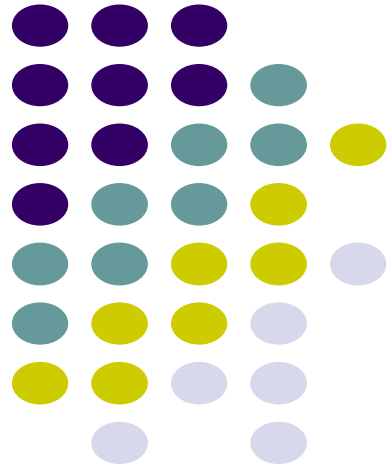
UNIVERSIDADE FEDERAL RURAL DO SEMI-ÁRIDO
DEPARTAMENTO DE CIÊNCIAS EXATAS E NATURAIS
CURSO DE CIÊNCIA DA COMPUTAÇÃO

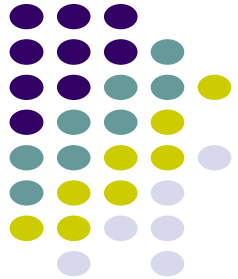
Arquitetura e Organização de Computadores

Melhorando o Desempenho com Pipeline - Parte II



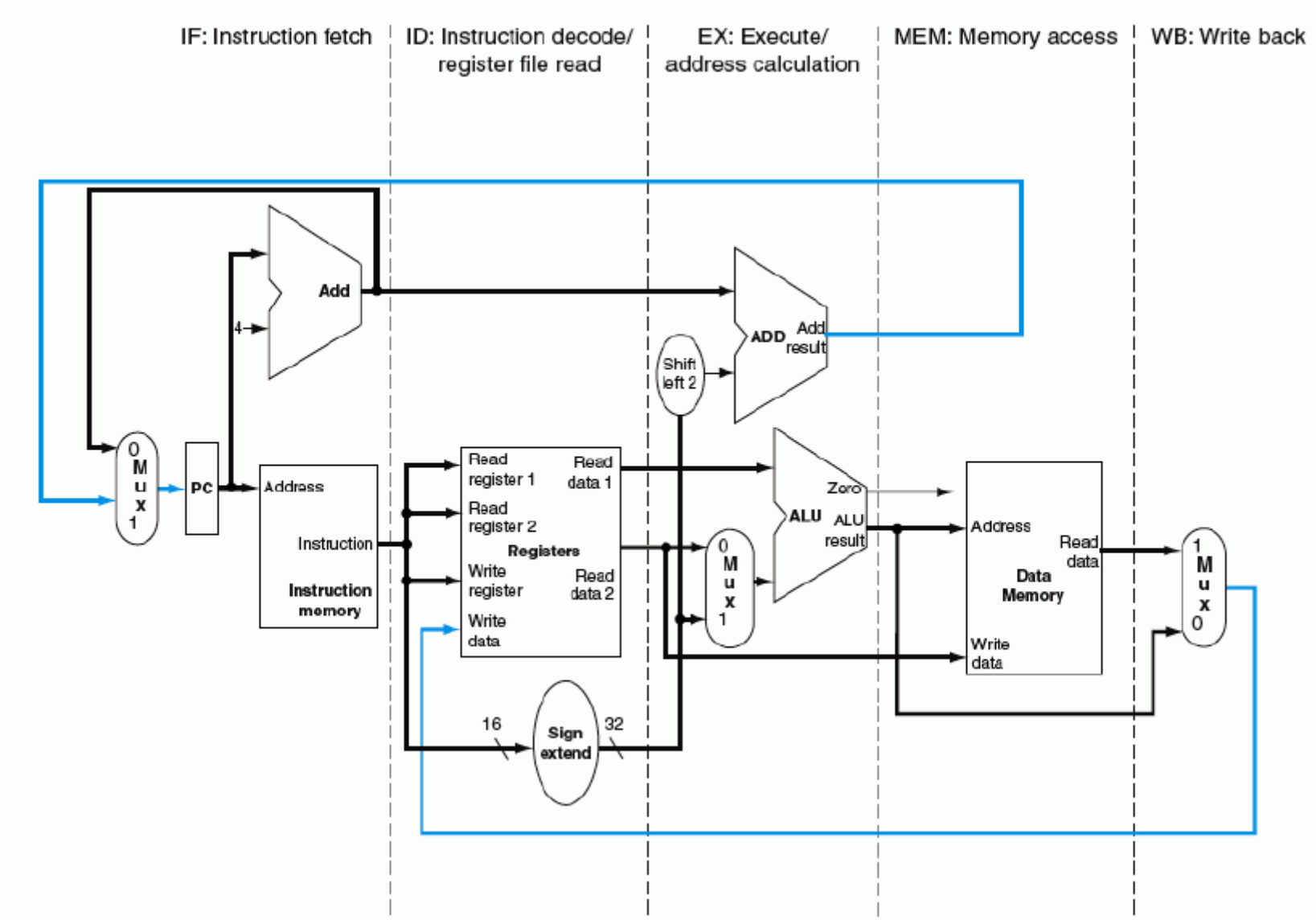
Prof. Sílvio Fernandes

Um caminho de dados usando pipeline

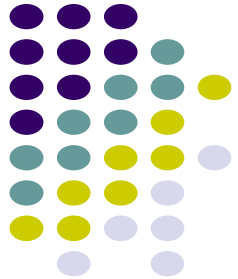


- A divisão de uma instrução em 5 estágios significa um pipeline de 5 estágio, que, significa que até 5 instruções estarão em execução durante qualquer ciclo de clock
- Cada parte possuindo um nome correspondente a um estágio da execução:
  1. IF (Instruction Fetch): Busca de instruções
  2. ID (Instruction Decode): Decodificação de instruções e leitura do banco de registradores
  3. EX: Execução ou cálculo de endereço
  4. MEM: Acesso à memória de dados
  5. WB (Write Back): Escrita do resultado

Um caminho de dados usando pipeline



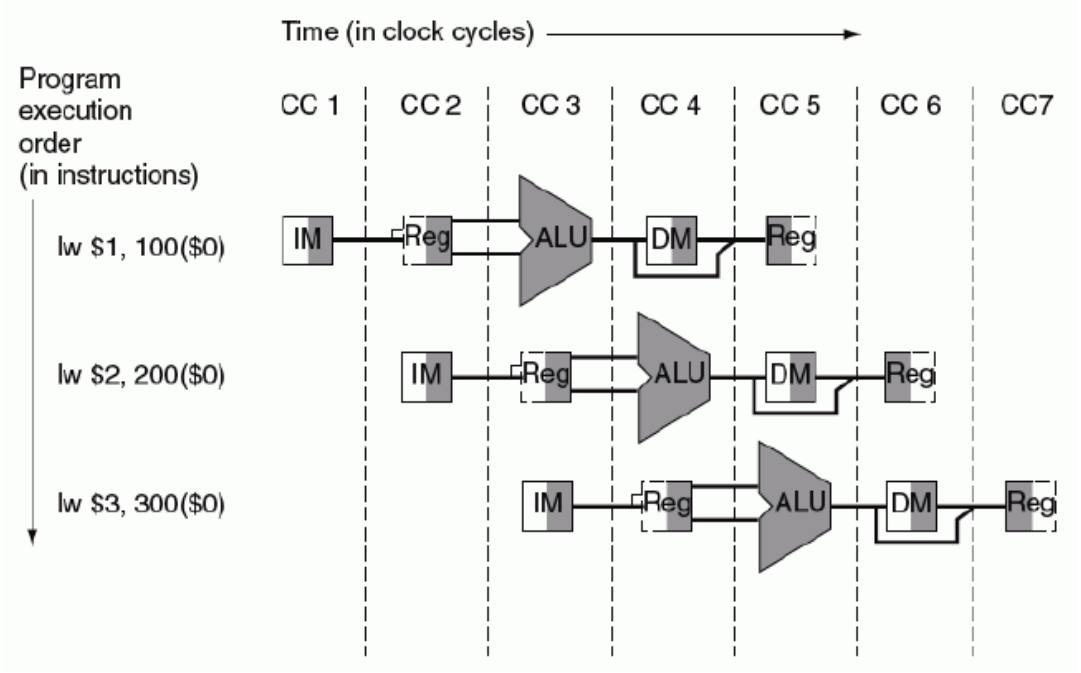
Um caminho de dados usando pipeline



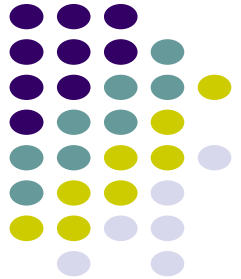
- Esses 5 componentes correspondem aproximadamente ao modo como o caminho de dados é desenhado
  - As instruções e os dados em geral se movem da esquerda para a direita pelos 5 estágios enquanto completam a execução
  - Exceções:
    - O estágio de escrita do resultado, que coloca o resultado de volta no banco de registradores, no meio do caminho de dados
    - A seleção do próximo valor do PC, escolhendo entre o PC incrementado e o endereço de desvio do estágio MEM

Um caminho de dados usando pipeline

- Uma maneira de mostrar o que acontece na execução com pipeline é fingir que cada instrução tem seu próprio caminho de dados, e depois colocar esses caminhos em uma linha de tempo para mostrar seu relacionamento

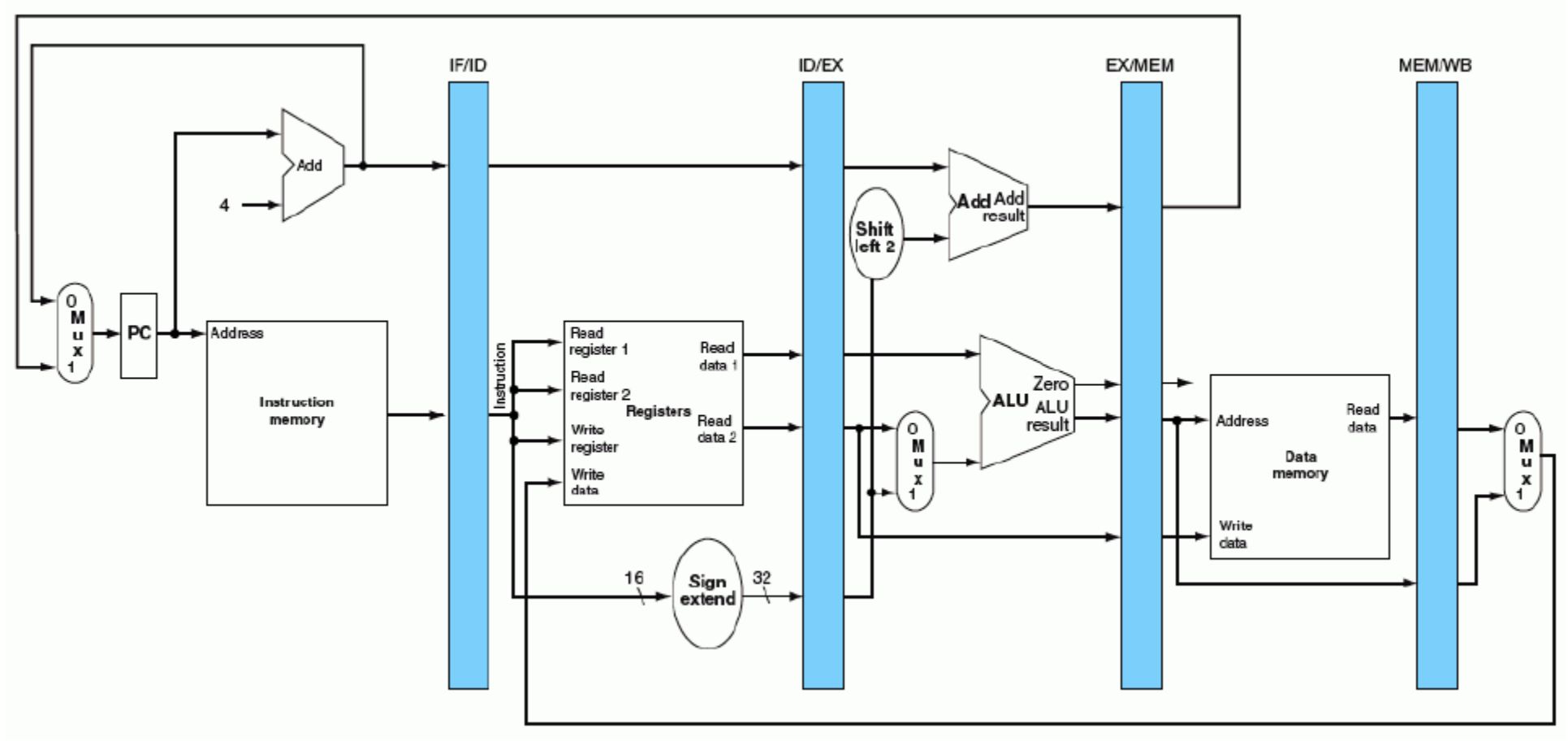


Um caminho de dados usando pipeline

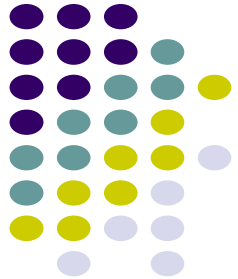


- Assim como precisamos adicionar novos registradores para armazenar valores intermediários no caminho de dados multiciclo, podemos adicionar registradores entre os estágios do pipeline
- Todas as instruções avançam durante cada ciclo de um registrador do pipeline para o seguinte
- Os registradores recebem os nome dos 2 estágios separador por esse registrador

Um caminho de dados usando pipeline

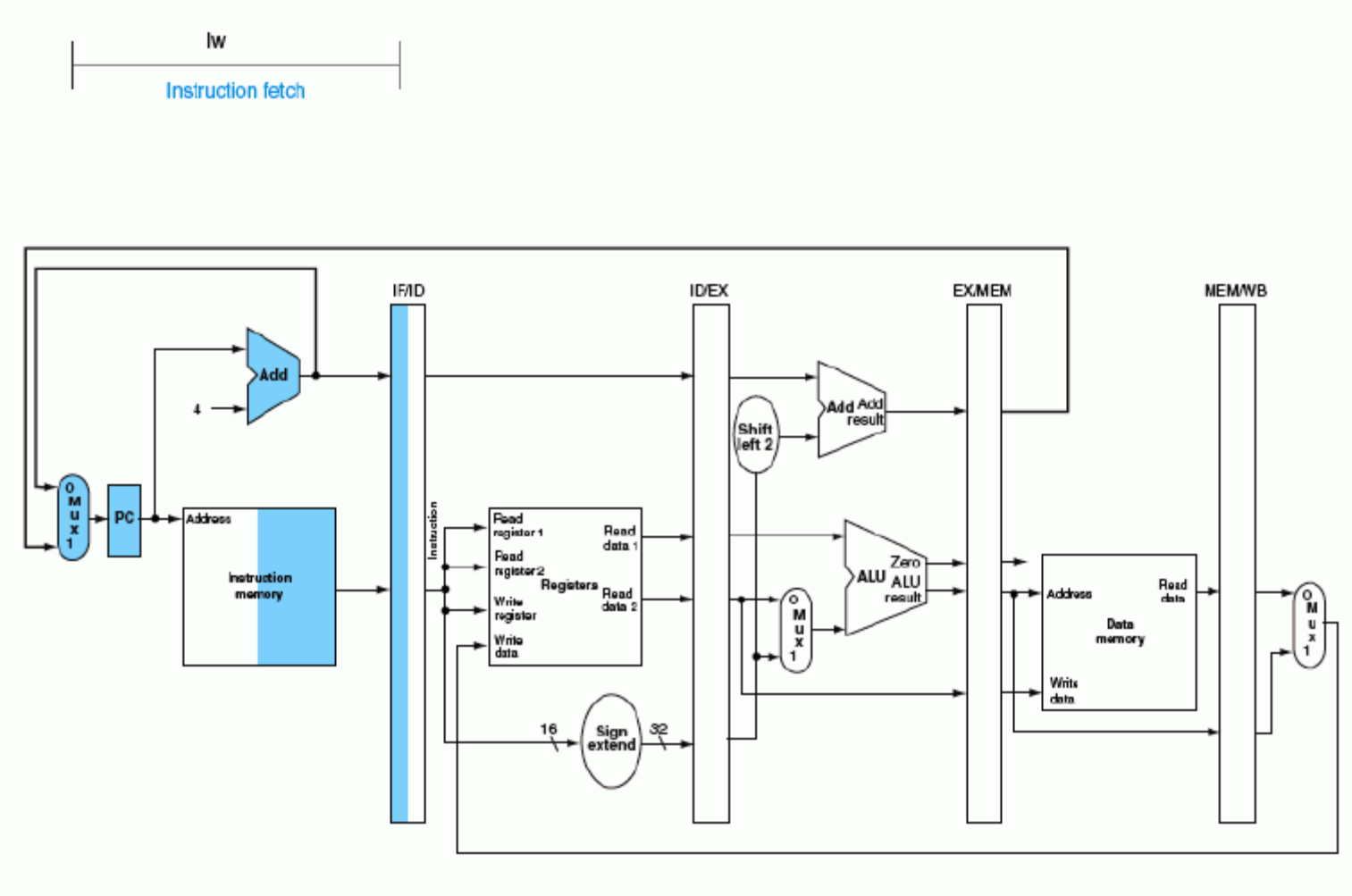


Um caminho de dados usando pipeline

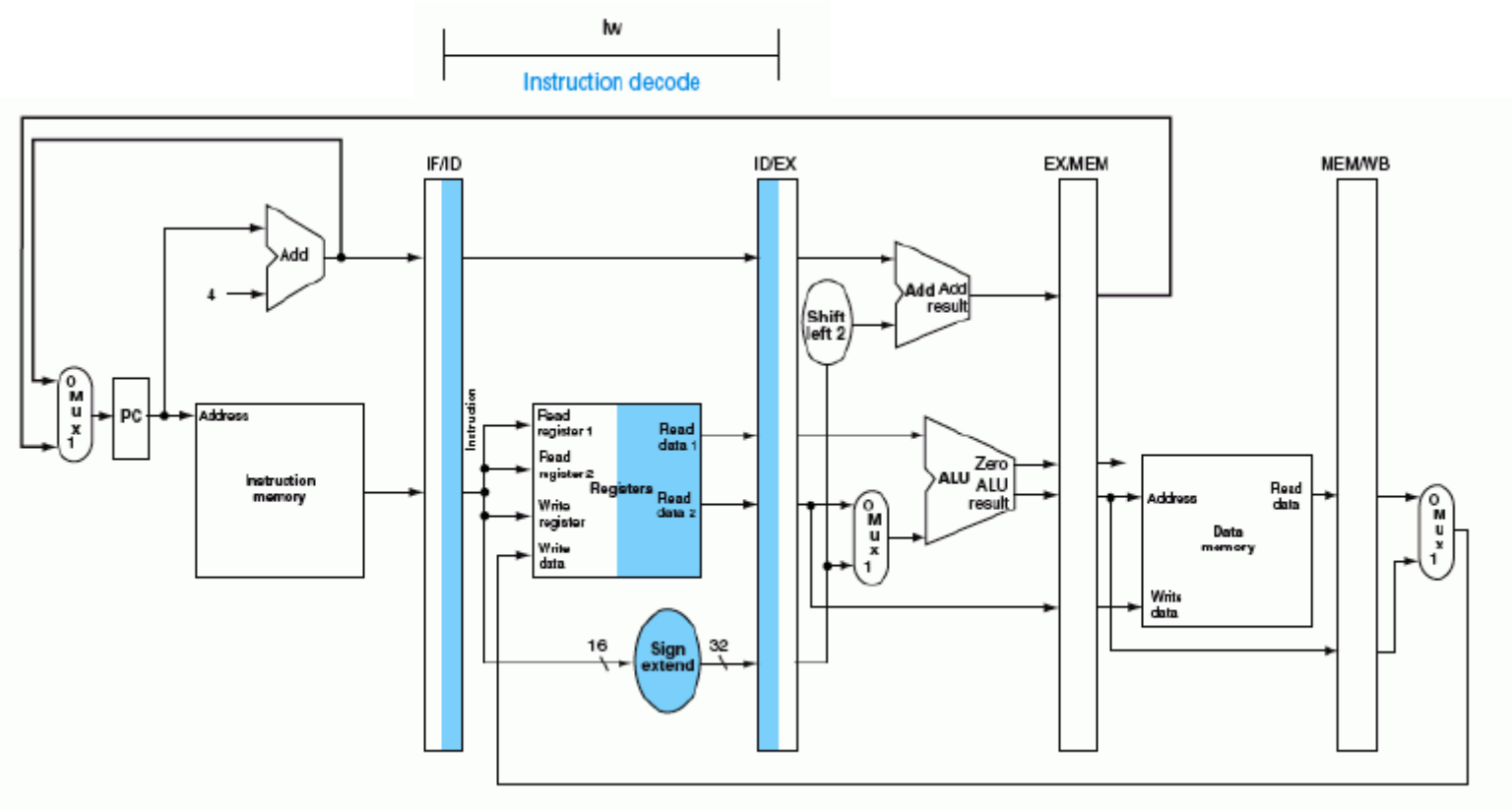


- O PC pode ser considerado um registrador de pipeline: um que alimenta o estágio IF do pipeline
- O PC faz parte do estado arquitetônico visível; seu conteúdo precisa ser salvo quando ocorre uma exceção, enquanto o conteúdo dos registradores de pipeline pode ser descartado
- Para mostrar como funciona a técnica de pipelining, apresentaremos sequências de figuras para demonstrar a operação com o tempo

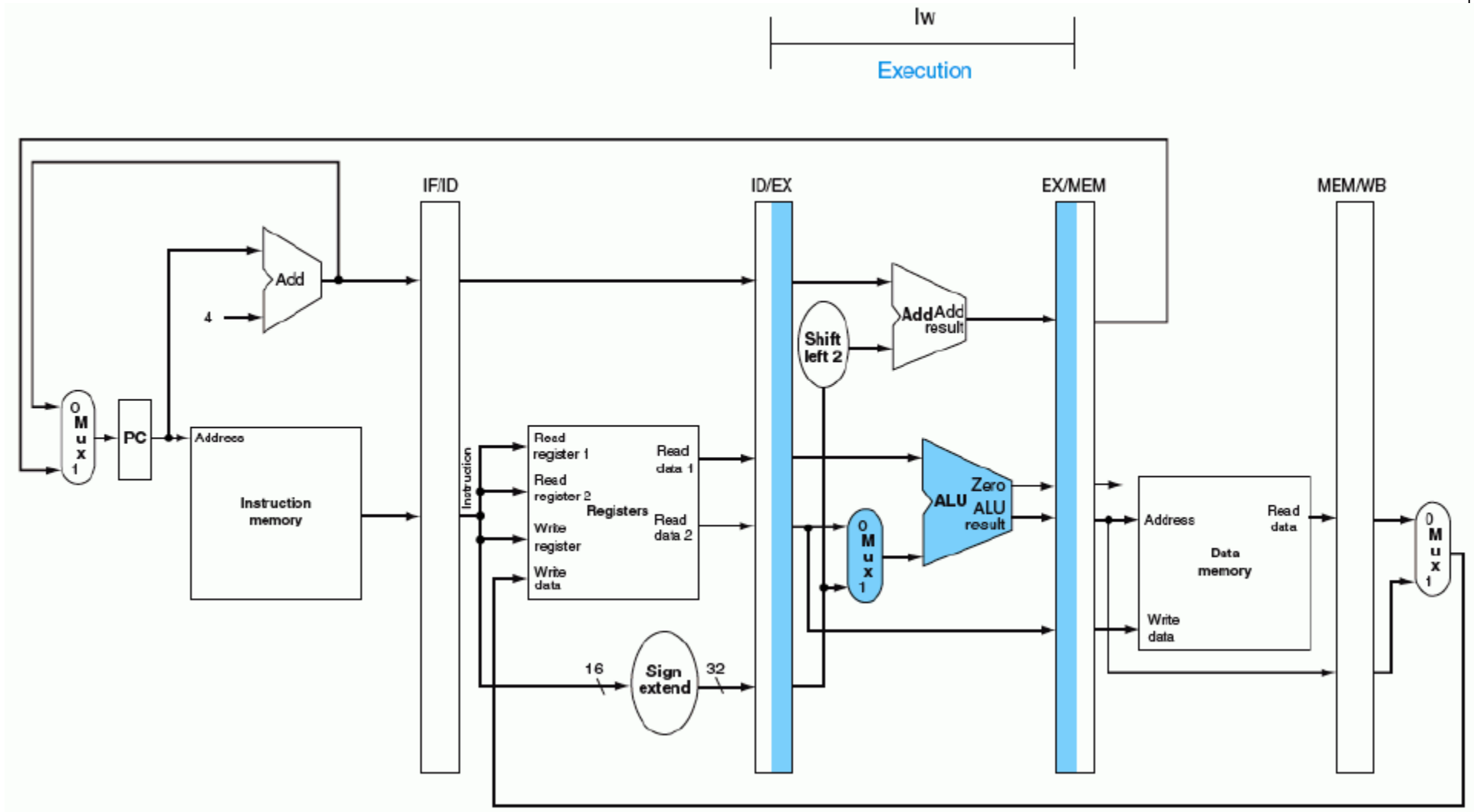
Um caminho de dados usando pipeline



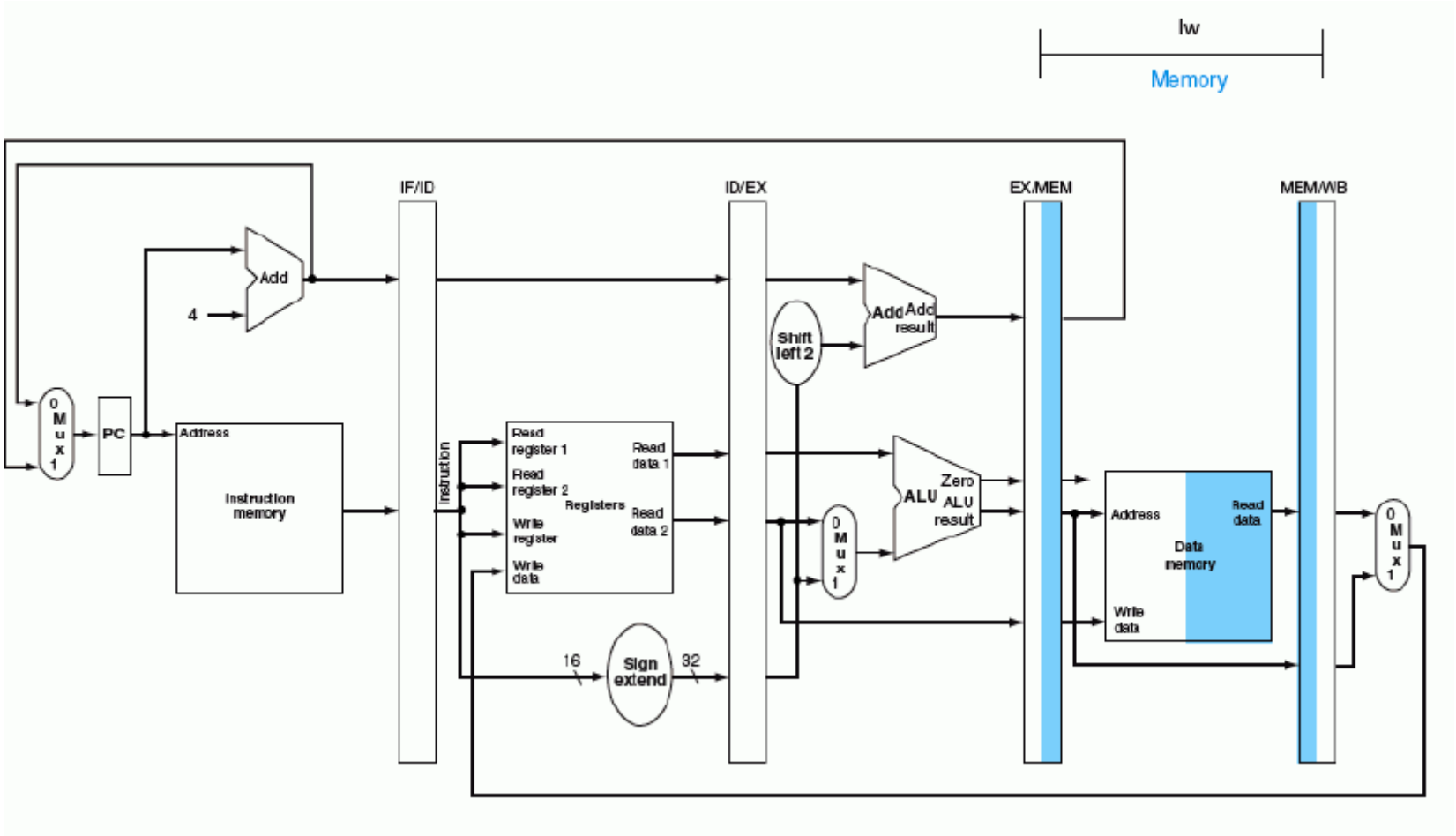
Um caminho de dados usando pipeline



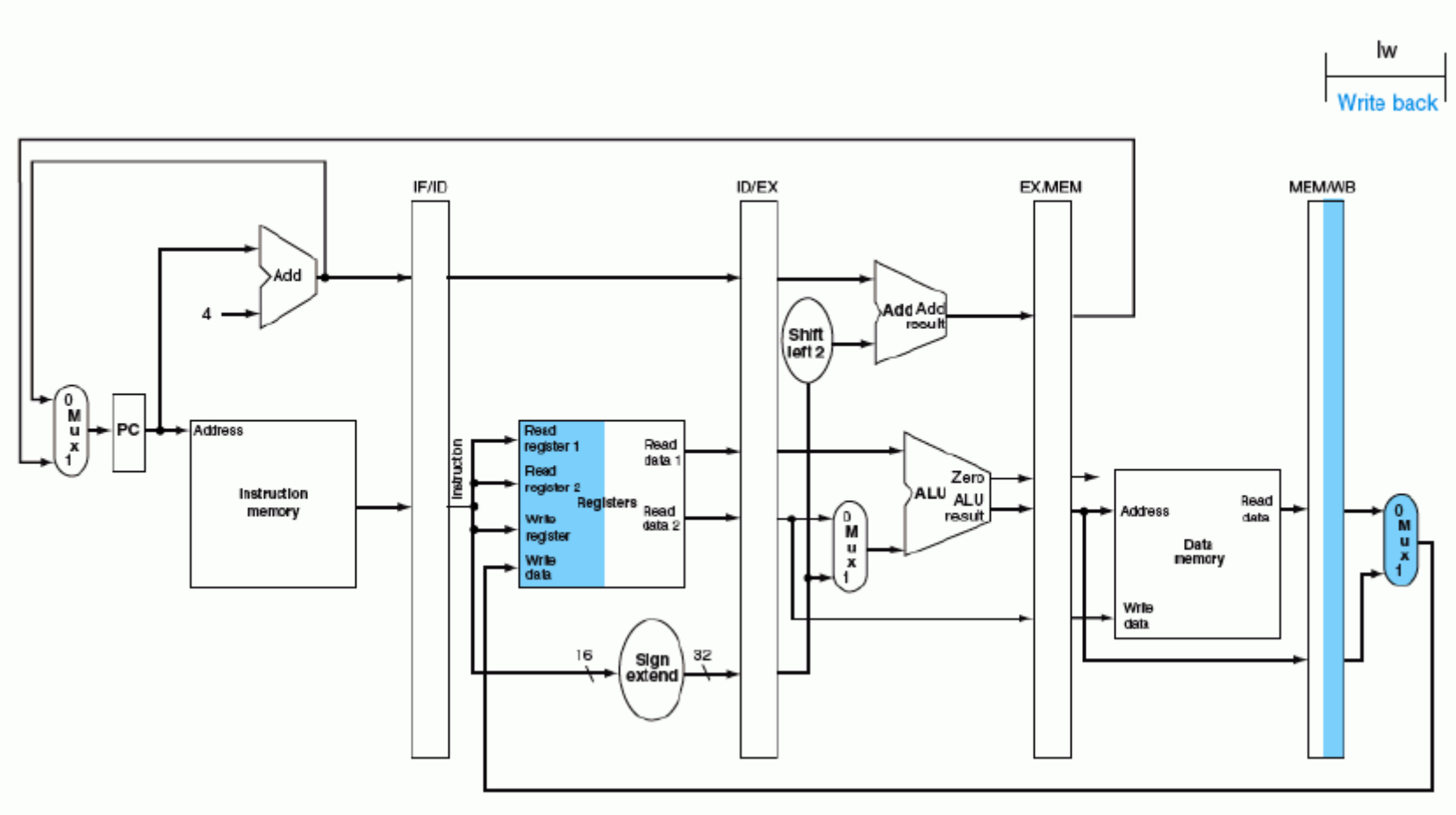
Um caminho de dados usando pipeline



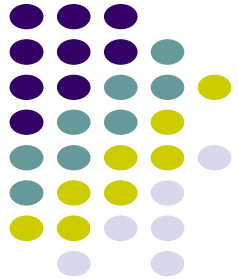
Um caminho de dados usando pipeline



Um caminho de dados usando pipeline

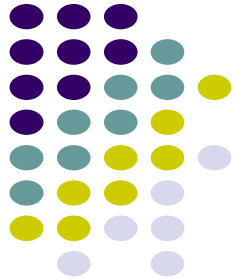


Um caminho de dados usando pipeline



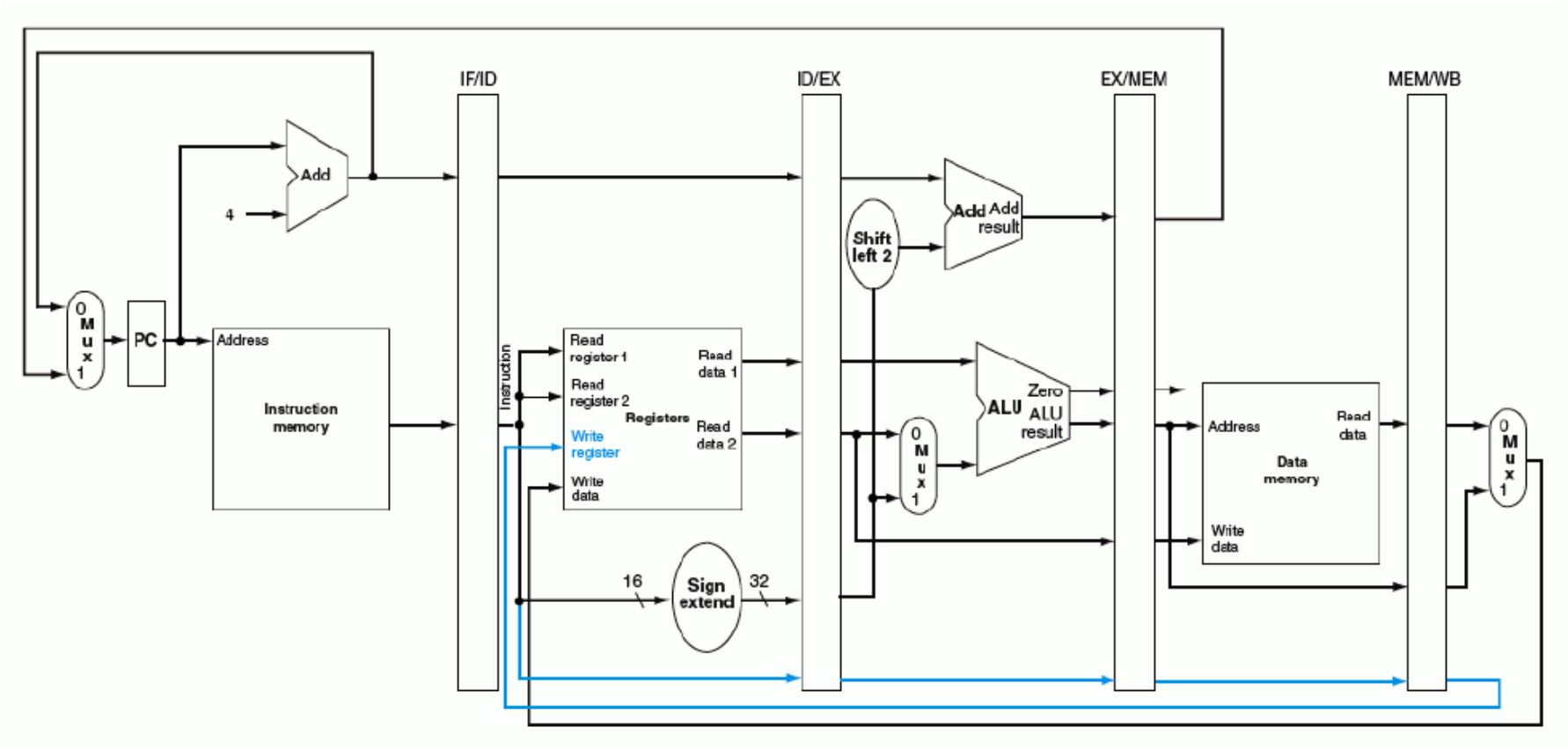
- Cada componente lógico do caminho de dados – como memória de instruções, portas para leitura de registradores, ALU, memória de dados e porta para escrita de registradores – só pode ser usado dentro de um único estágio do pipeline
- Caso contrário, teríamos um harzard estrutural
- Logo, esses componentes e seu controle podem ser associados a um único estágio do pipeline

Um caminho de dados usando pipeline



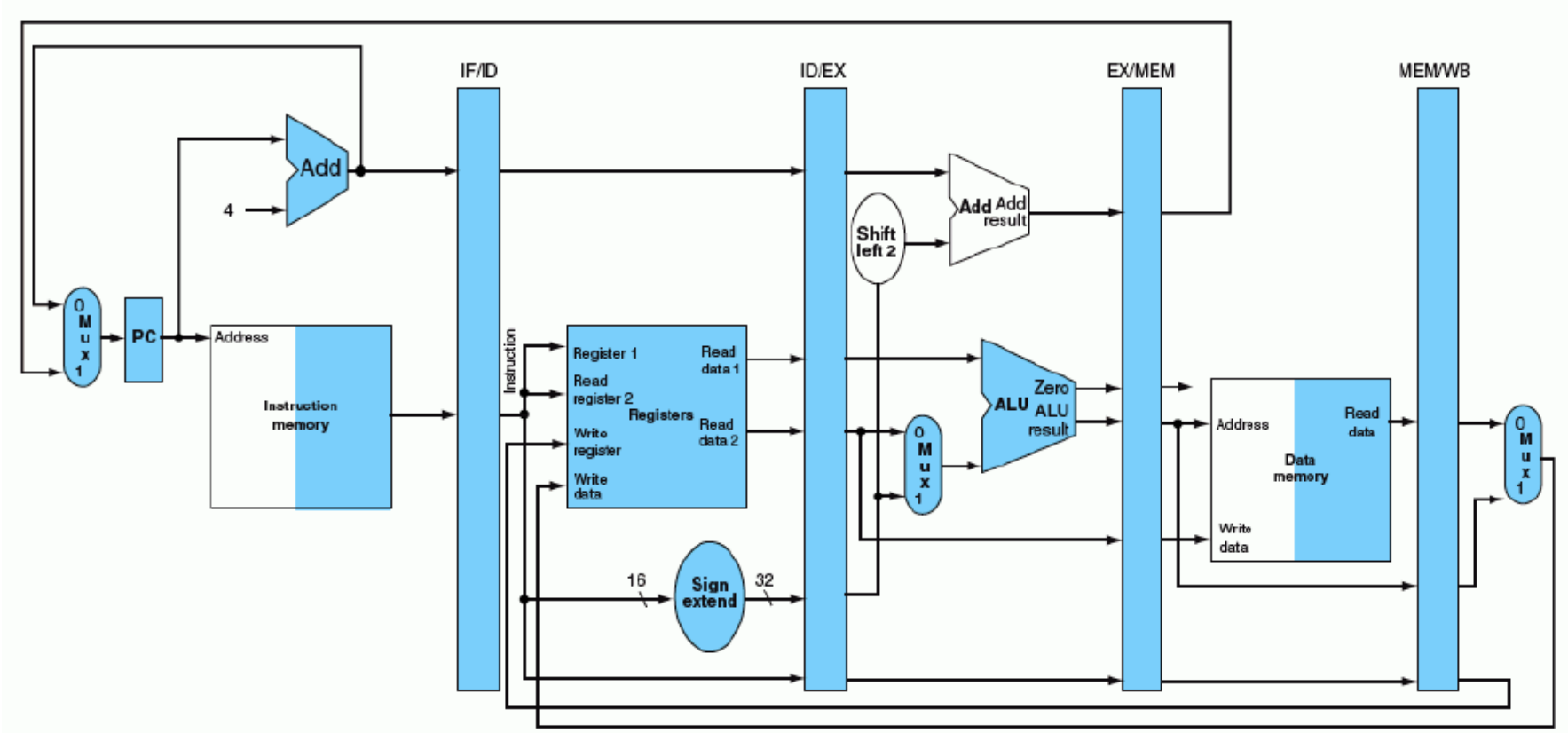
- Agora, podemos descobrir um bug no projeto da instrução load
  - Qual estágio fornece o número do registrador de escrita?
    - A instrução no registrador IF/ID fornece o número do registrador de escrita, embora essa instrução ocorra consideravelmente depois da instrução load!
  - Logo, precisamos preservar o número do registrador de destino da instrução load
    - Precisa passar o número do registrador ID/EX por EX/MEM para o registador MEM/WB, para uso no estágio WB

Um caminho de dados usando pipeline

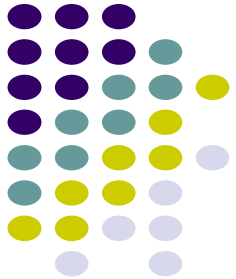


Um caminho de dados usando pipeline

- Caminho de dados usado em todos os 5 estágios de uma instrução load



Um caminho de dados usando pipeline



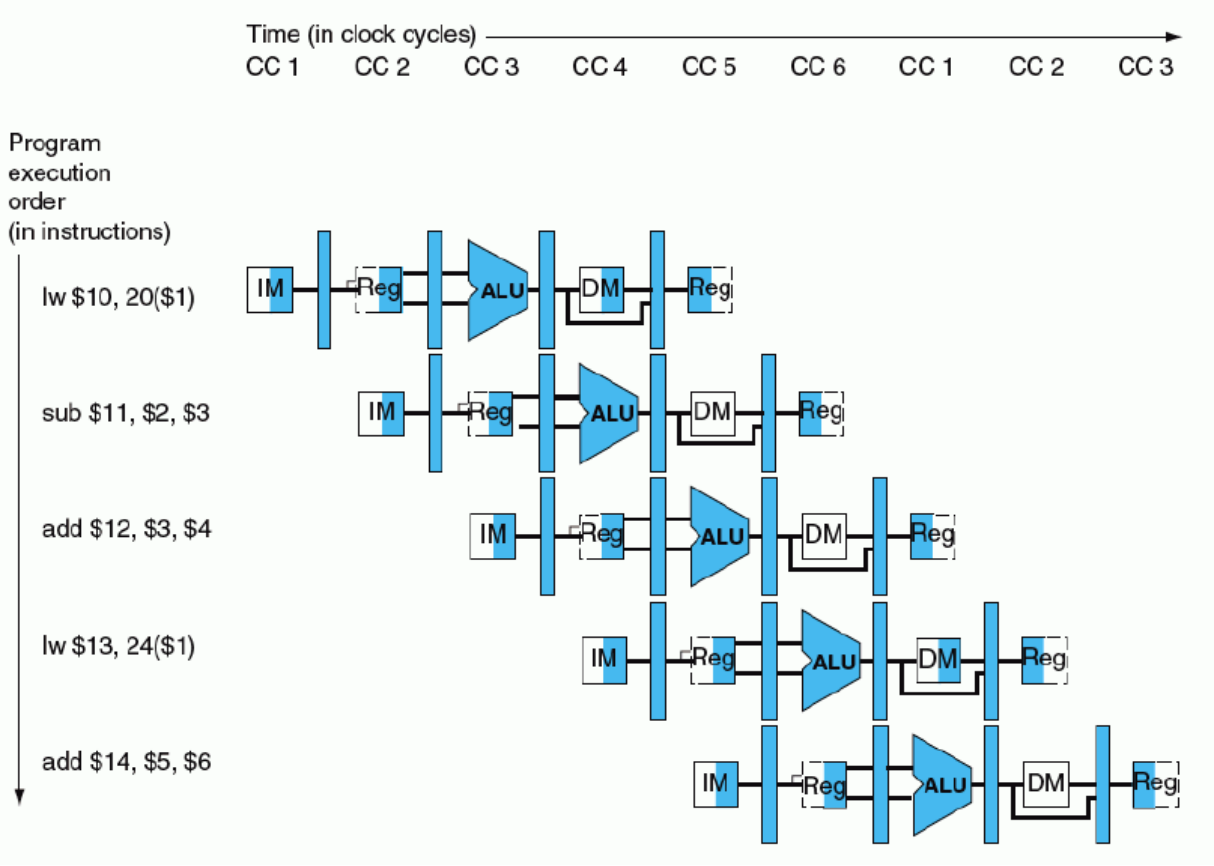
- Representando pipelines graficamente
  - Pode ser feita através de:
    - *diagrama de pipeline com múltiplos ciclos de clock*
    - *diagrama de pipeline com único ciclo de clock*
  - Considere a seguinte sequência de instruções:

```
lw      $10, 20($1)
sub     $11, $2, $3
add     $12, $3, $4
lw      $13, 24($1)
add     $14, $5, $6
```

  - Veja as representações a seguir

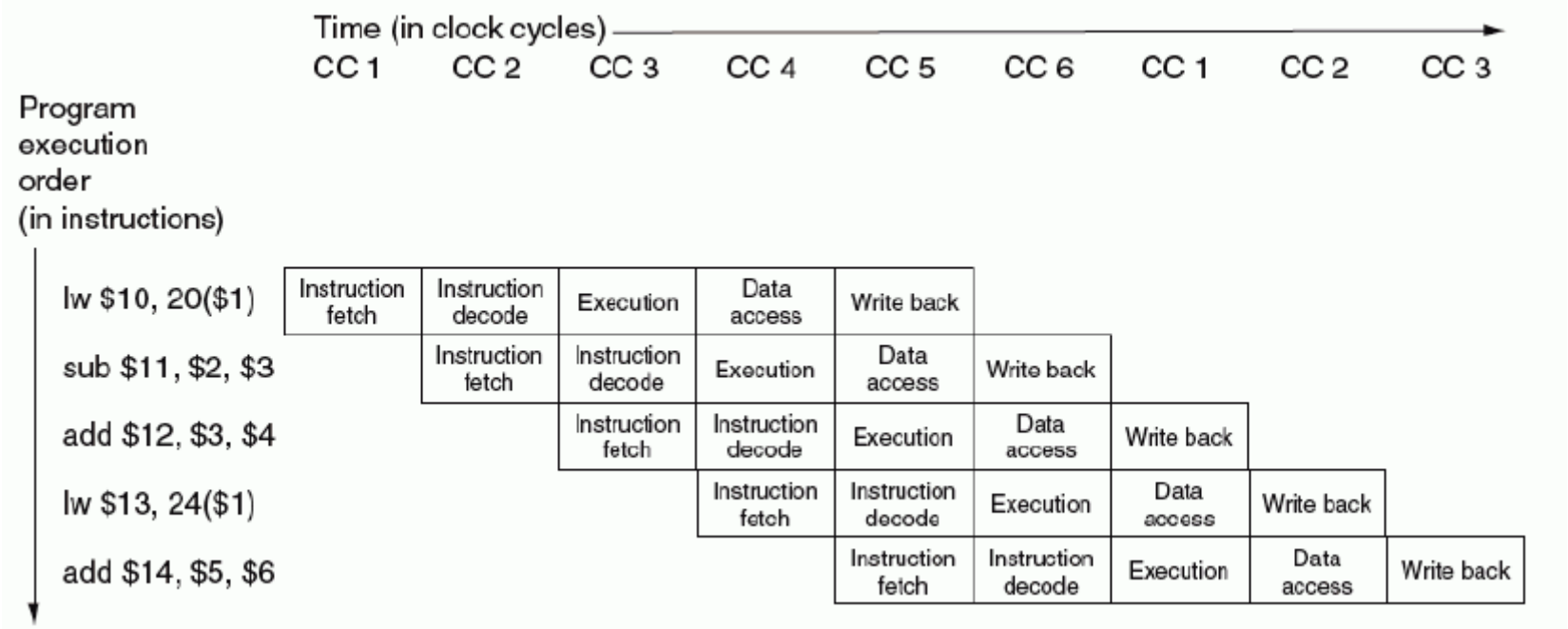
Um caminho de dados usando pipeline

- Representando pipelines graficamente
  - *Diagrama de pipeline com múltiplos ciclos de clock*

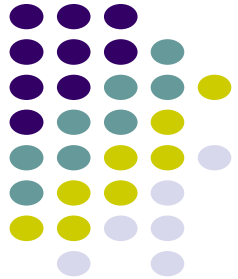


Um caminho de dados usando pipeline

- Representando pipelines graficamente
  - *Diagrama de pipeline com múltiplos ciclos de clock tradicional*



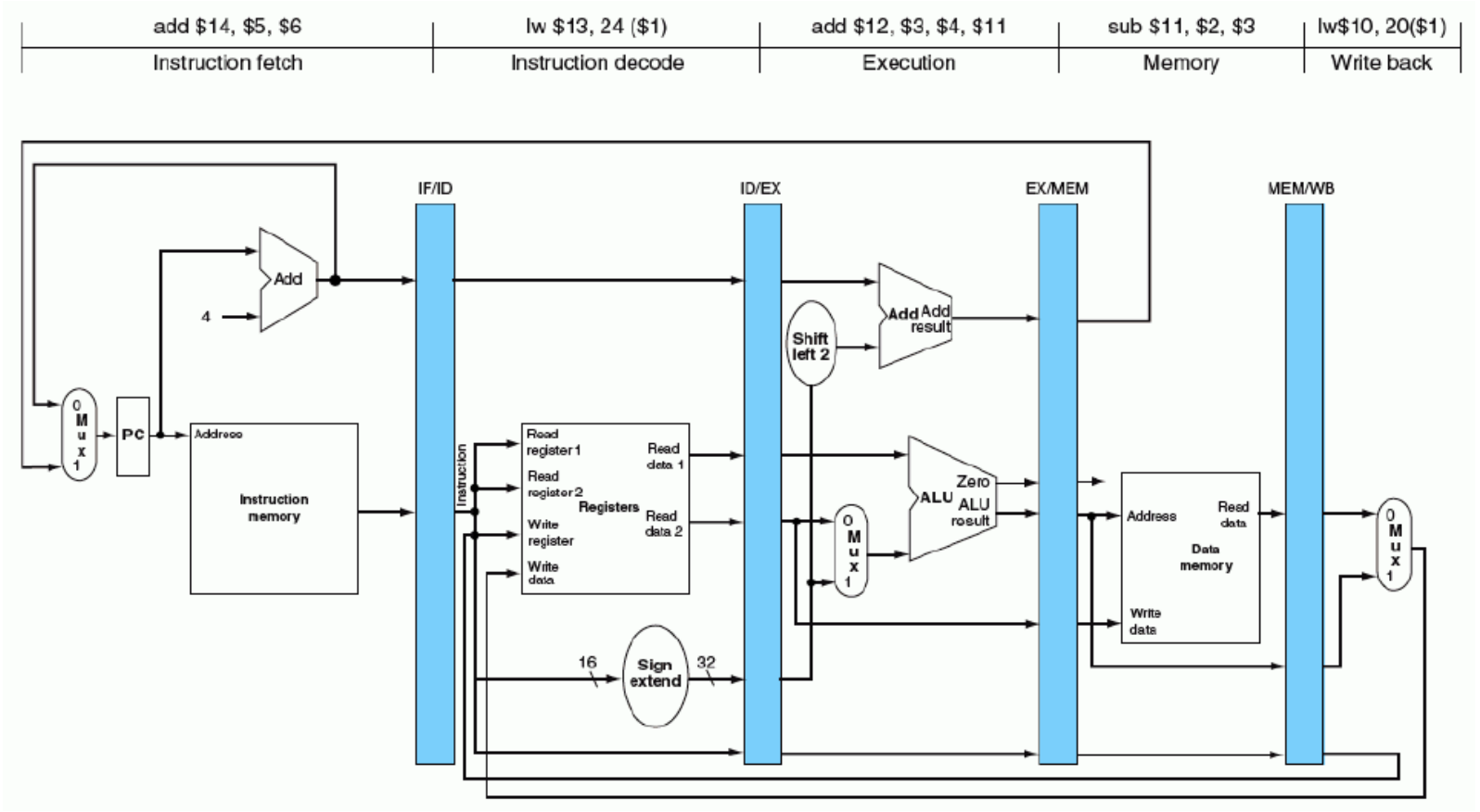
Um caminho de dados usando pipeline



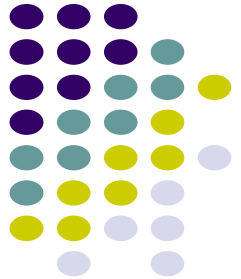
- Representando pipelines graficamente
  - *Diagrama de pipeline com único ciclo de clock*
    - Mostram o estado do caminho de dados inteiro durante um único ciclo, e normalmente todas as 5 instruções são identificadas por rótulos acima de seus respectivos estágios do pipeline

Um caminho de dados usando pipeline

- Representando pipelines graficamente
  - *Diagrama de pipeline com único ciclo de clock*

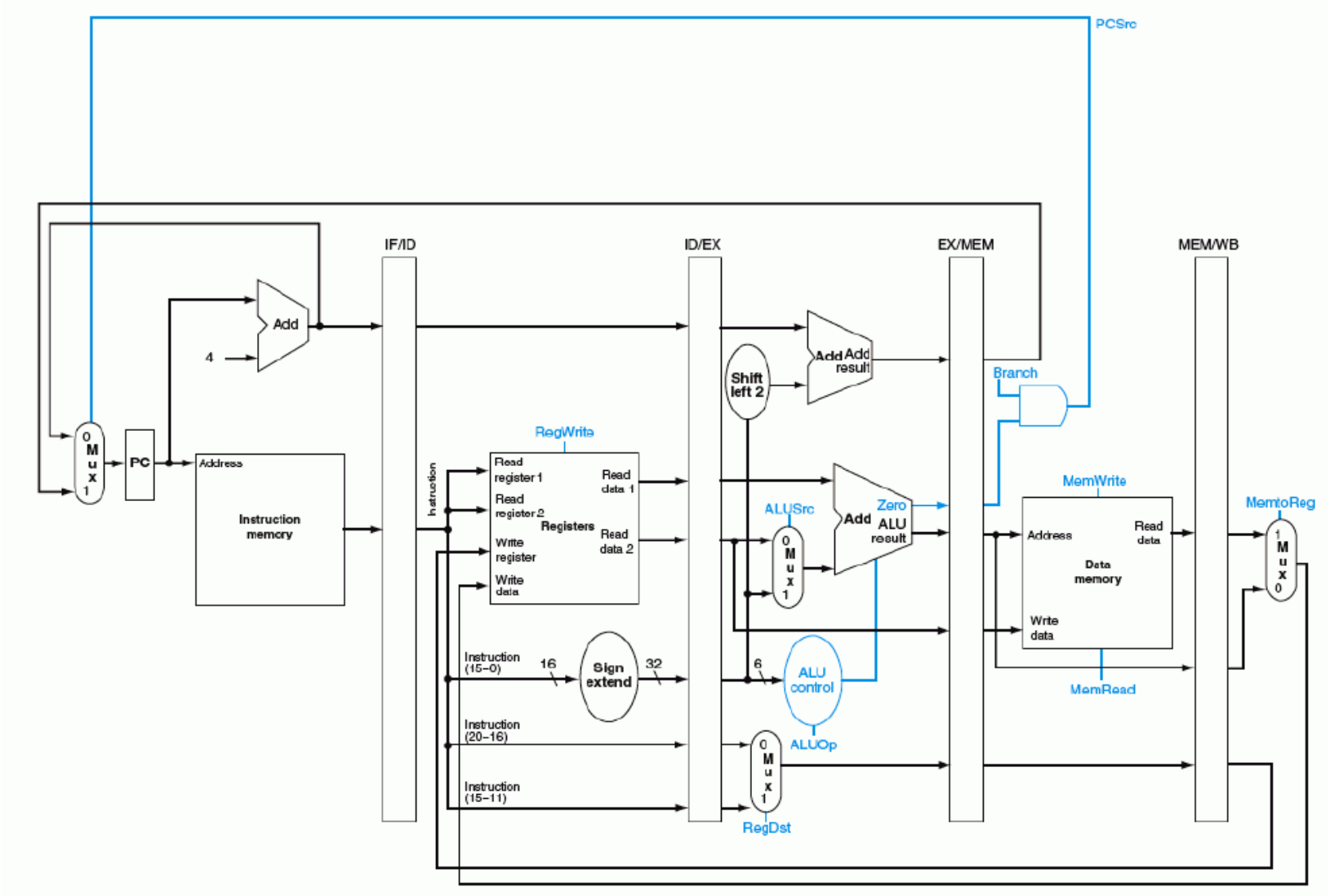


Controle de um pipeline

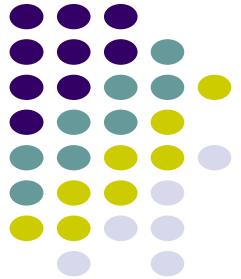


- O 1º passo é rotular as linhas de controle no caminho de dados existente
  - Pegamos o máximo possível emprestado do controle para o caminho de dados simples
  - Usamos a mesma lógica de controle da ALU, lógica de desvio, multiplexador do registrador destino e linhas de controle
  - Não existem sinais de escrita para os registradores de pipeline (IF/ID, ID/EX, EX/MEM e MEM/WB), pois eles são escrito durante cada ciclo de clock

Controle de um pipeline

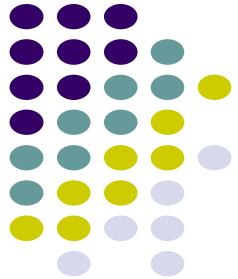


Controle de um pipeline



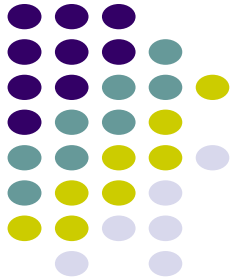
- Para especificar o controle para o pipeline, só precisamos definir os valores de controle durante cada estágio do pipeline
- Cada linha de controle está associada a um componente ativo em apenas um estágio do pipeline, podemos dividir as linhas de controle em 5 grupos, de acordo com o estágio do pipeline

Controle de um pipeline



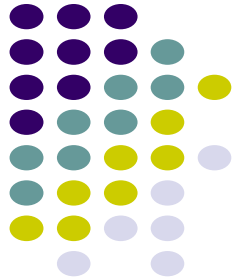
1. **Busca de instruções:** os sinais de controle para ler a memória de instruções e escrever o PC sempre são ativados, de modo que não existe nada de especial para controlar nesse estágios do pipeline
2. **Decodificação de instruções/leitura do banco de registradores:** como no estágio anterior, a mesma coisa acontece em cada ciclo de clock, de modo que não existem linhas de controle opcionais para definir

Controle de um pipeline



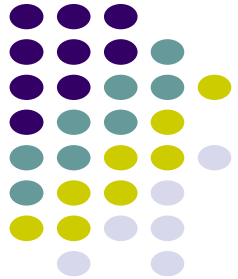
3. **Execução/cálculo de endereço:** os sinais a serem definidos são RegDst, ALUOp, e ALUSrc. Os sinais selecionam o registrador destino, a operação da ALU e Dados da leitura 2 ou um imediato com sinal estendido para a ALU
4. **Acesso à memória:** linhas de controle definidas nesse estágio são Branch, MemRead e MemWrite. Esses sinais são definidos pelas instruções branch equal, load e store, respectivamente.

Controle de um pipeline



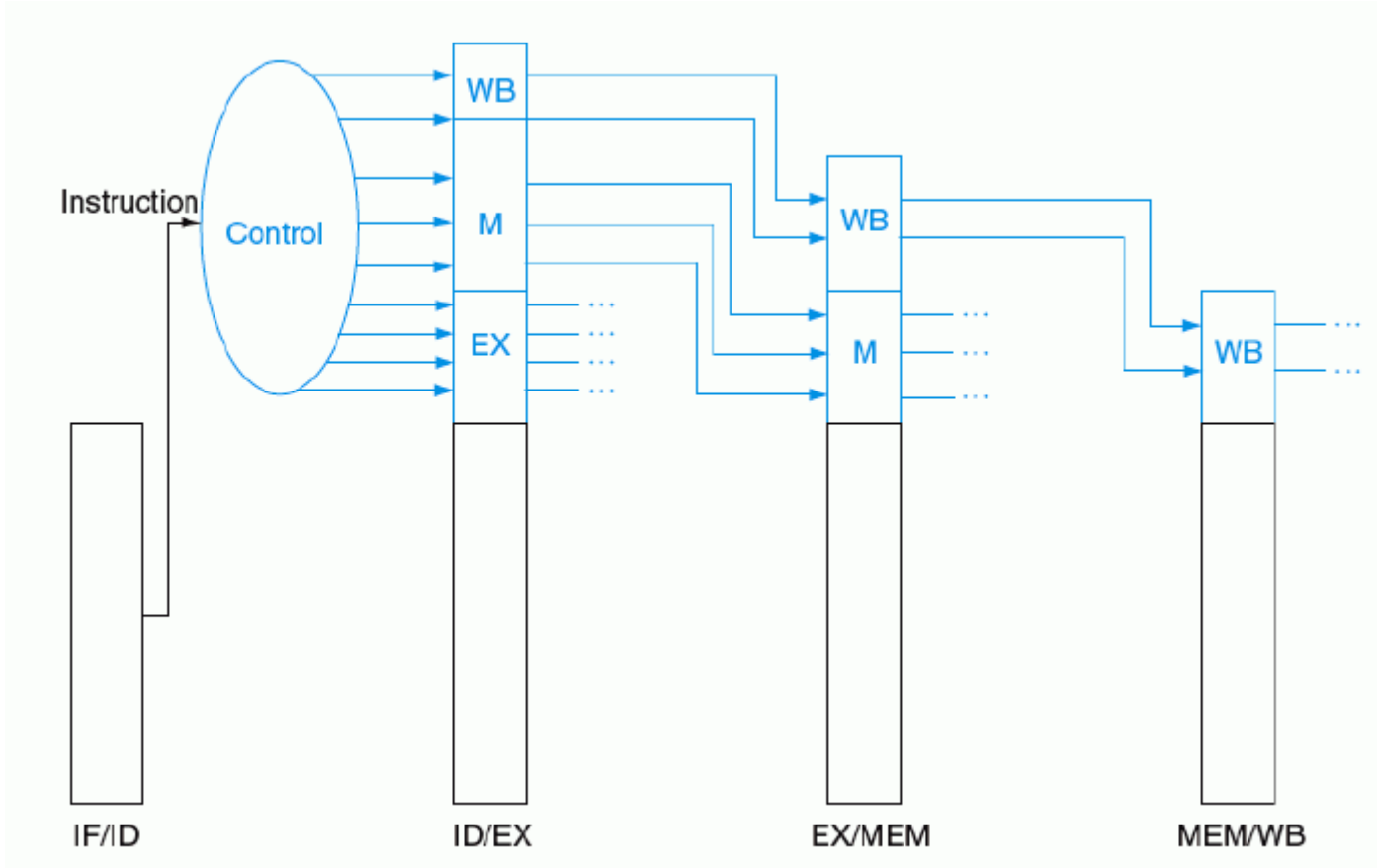
5. **Escrita do resultado:** as duas linhas de controle são MemtoReg, que decide entre enviar o resultado da ALU ou o valor da memória para o banco de registradores, e RegWrite, que escreve o valor escolhido

Controle de um pipeline

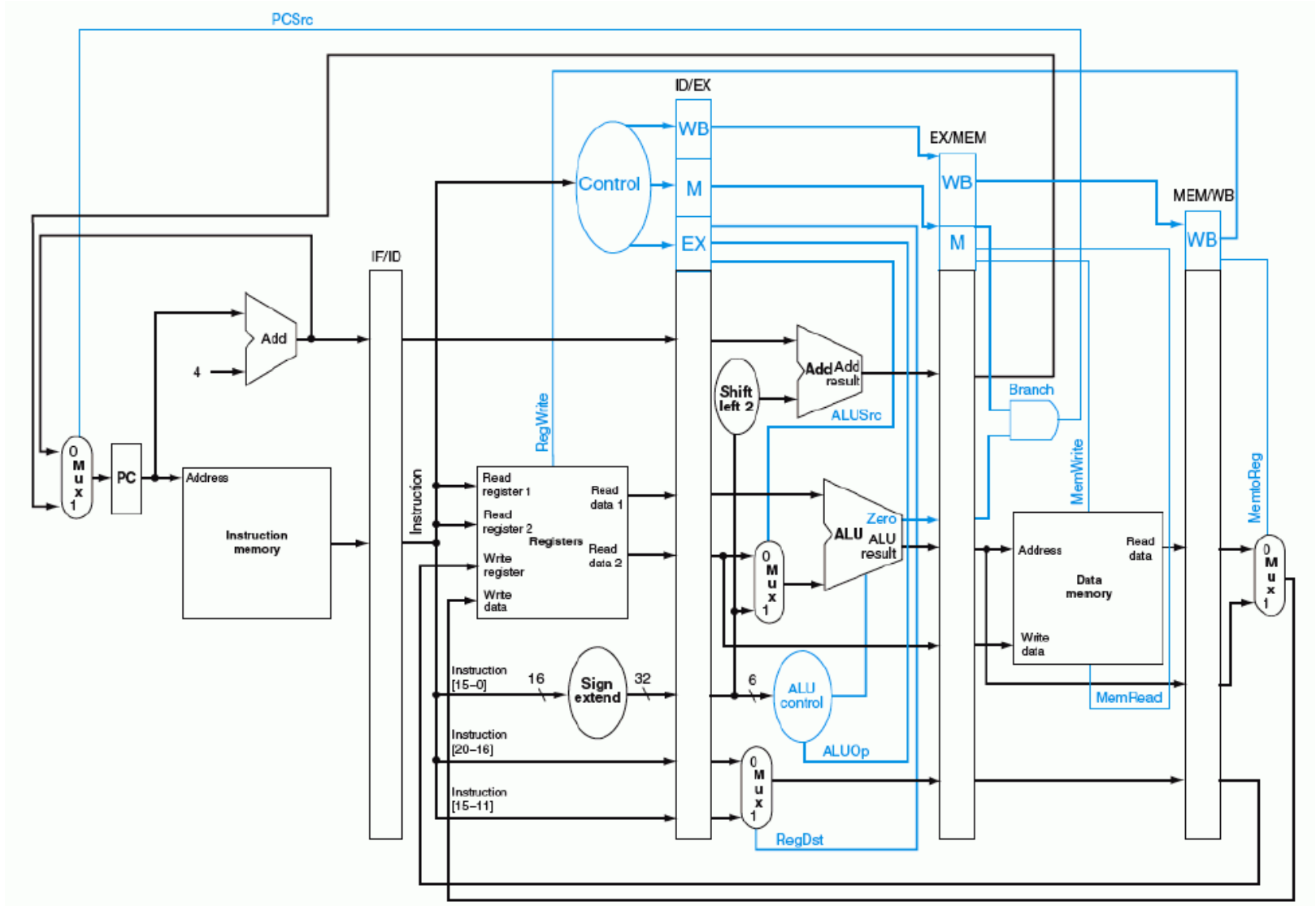


- Como a utilização de um pipeline o caminho de dados deixa inalterado o significado das linhas de controle
- As nove linhas de controle estão agrupadas por estágio
- A implementação do controle significa definir as nove linhas para cada estágio, para cada instrução
  - A maneira mais simples de fazer isso é estender os registradores do pipeline para incluir informações de controle

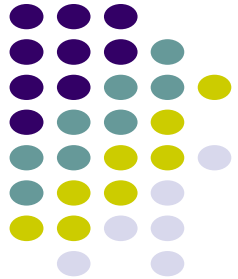
Controle de um pipeline



Controle de um pipeline

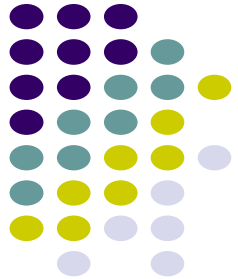


Controle de um pipeline



- Exemplo
  - Simulador Hades (Fig. 4.51)
    - Ex 1: Sem hazard
    - Ex 2: Hazard EXE
    - Ex 3: Hazard MEM

Hazards de dados: forwarding vs stalls

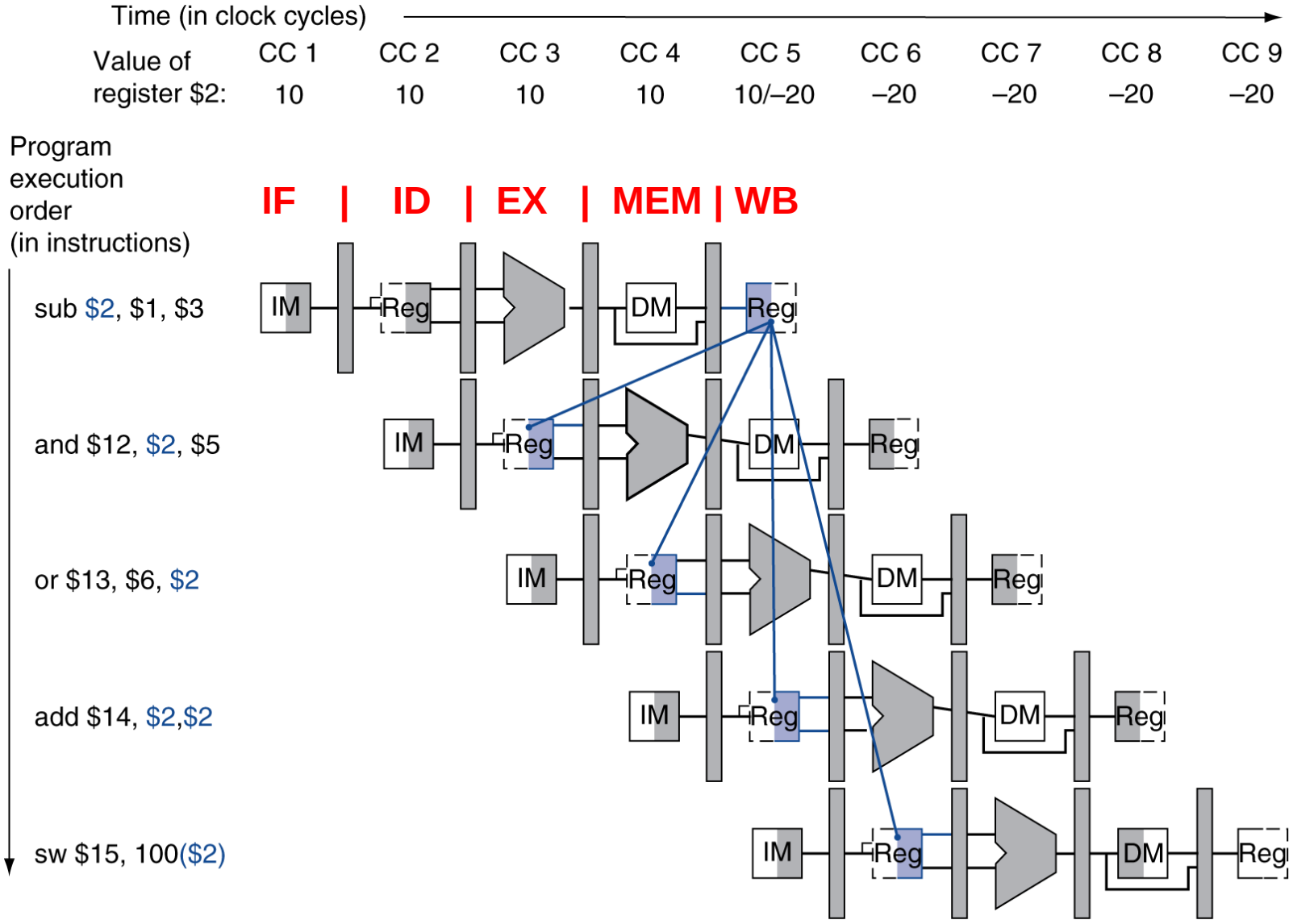


- Considere a sequência

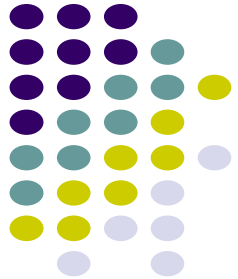
```
sub $2, $1,$3
and $12,$2,$5
or  $13,$6,$2
add $14,$2,$2
sw  $15,100($2)
```

- Como resolver as dependências com forwarding?
  - Como detectar?

Hazards de dados: forwarding vs stalls



Hazards de dados: forwarding vs stalls



- Passa o número do registrador ao longo do pipeline
  - ID/EX.RegisterRs = register number for Rs sitting in ID/EX pipeline register
- O número do registrador operando ALU em EX é dado por
  - ID/EX.RegisterRs, ID/EX.RegisterRt
- Hazard de dados podem ser

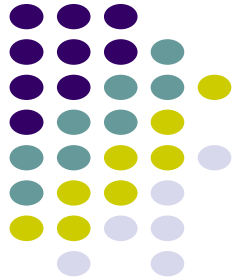
  1a. EX/MEM.RegisterRd = ID/EX.RegisterRs
  1b. EX/MEM.RegisterRd = ID/EX.RegisterRt

  Fwd de EX/MEM

  2a. MEM/WB.RegisterRd = ID/EX.RegisterRs
  2b. MEM/WB.RegisterRd = ID/EX.RegisterRt

  Fwd de MEM/WB

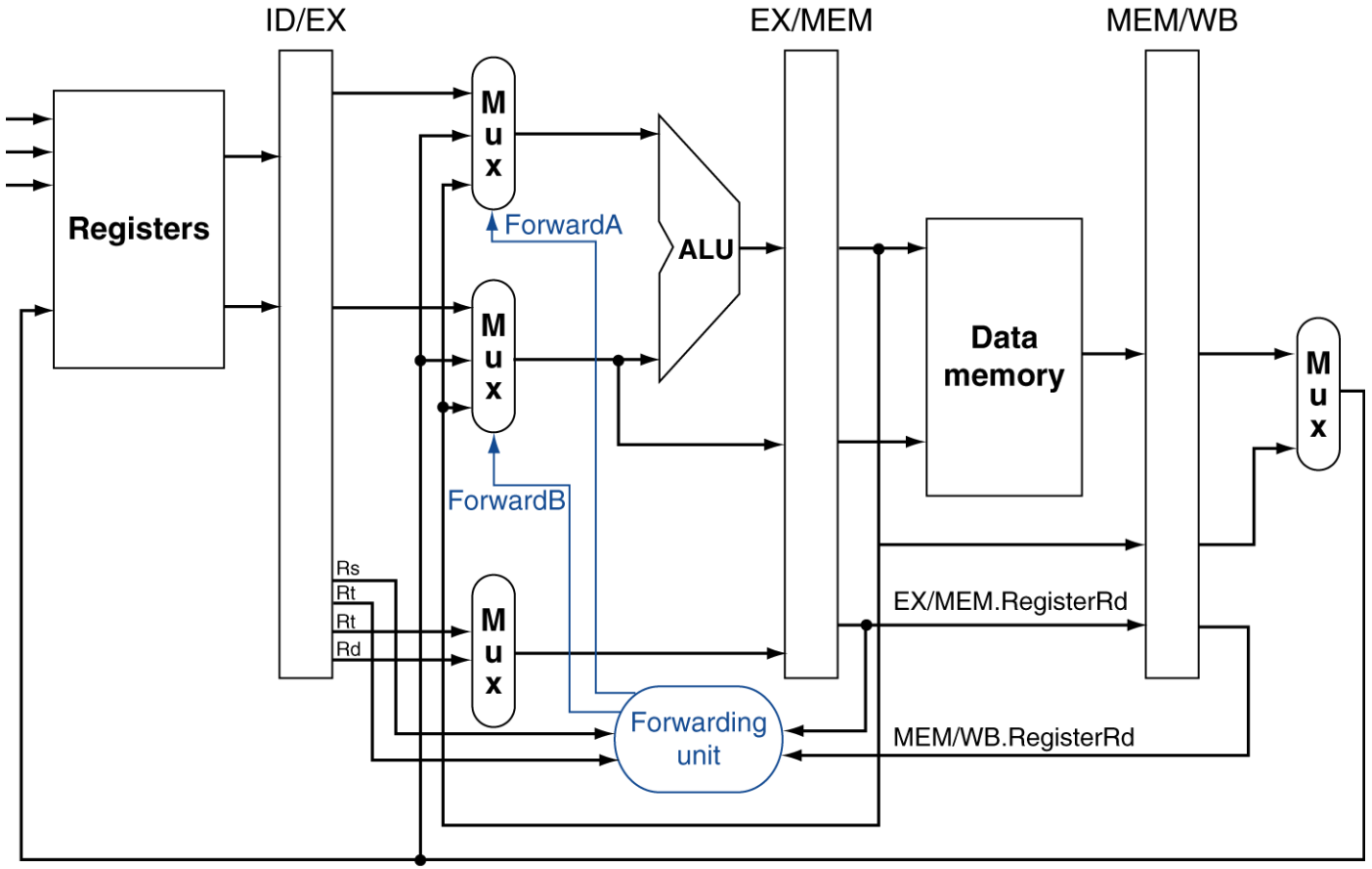
Hazards de dados: forwarding vs stalls



- Detectando a necessidade de forward
  - Apenas quando for escrever em registrador
    - EX/MEM.RegWrite, MEM/WB.RegWrite
  - E se apenas Rd for diferente de $zero
    - EX/MEM.RegisterRd ≠ 0, MEM/WB.RegisterRd ≠ 0

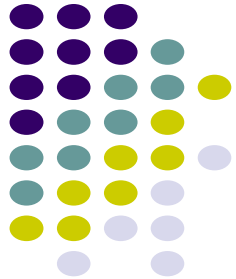
Hazards de dados: forwarding vs stalls

- Caminho de dados do forward



b. With forwarding

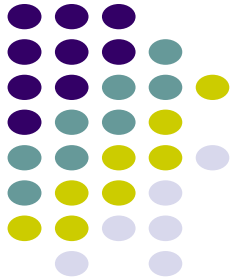
Hazards de dados: forwarding vs stalls



- Caminho de dados do forward
  - Condições do novos muxes

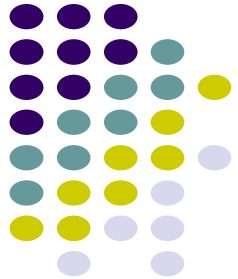
Mux control	Source	Explanation
ForwardA = 00	ID/EX	The first ALU operand comes from the register file.
ForwardA = 10	EX/MEM	The first ALU operand is forwarded from the prior ALU result.
ForwardA = 01	MEM/WB	The first ALU operand is forwarded from data memory or an earlier ALU result.
ForwardB = 00	ID/EX	The second ALU operand comes from the register file.
ForwardB = 10	EX/MEM	The second ALU operand is forwarded from the prior ALU result.
ForwardB = 01	MEM/WB	The second ALU operand is forwarded from data memory or an earlier ALU result.

Hazards de dados: forwarding vs stalls



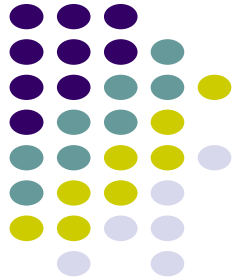
- Condições de Forward
- EX hazard
  - if (EX/MEM.RegWrite and (EX/MEM.RegisterRd ≠ 0)
    and (EX/MEM.RegisterRd = ID/EX.RegisterRs))
    ForwardA = 10
  - if (EX/MEM.RegWrite and (EX/MEM.RegisterRd ≠ 0)
    and (EX/MEM.RegisterRd = ID/EX.RegisterRt))
    ForwardB = 10
- MEM hazard
  - if (MEM/WB.RegWrite 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

Hazards de dados: forwarding vs stalls



- Exemplo
  - Simulador Hades (Fig. 4.56)
    - Ex 1: Sem hazard
    - Ex 2: Hazard EXE
    - Ex 3: Hazard MEM
    - Ex 4: Branch

Hazards de dados: forwarding vs stalls

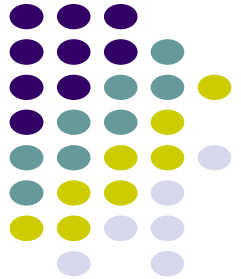


- Considere esta sequência

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

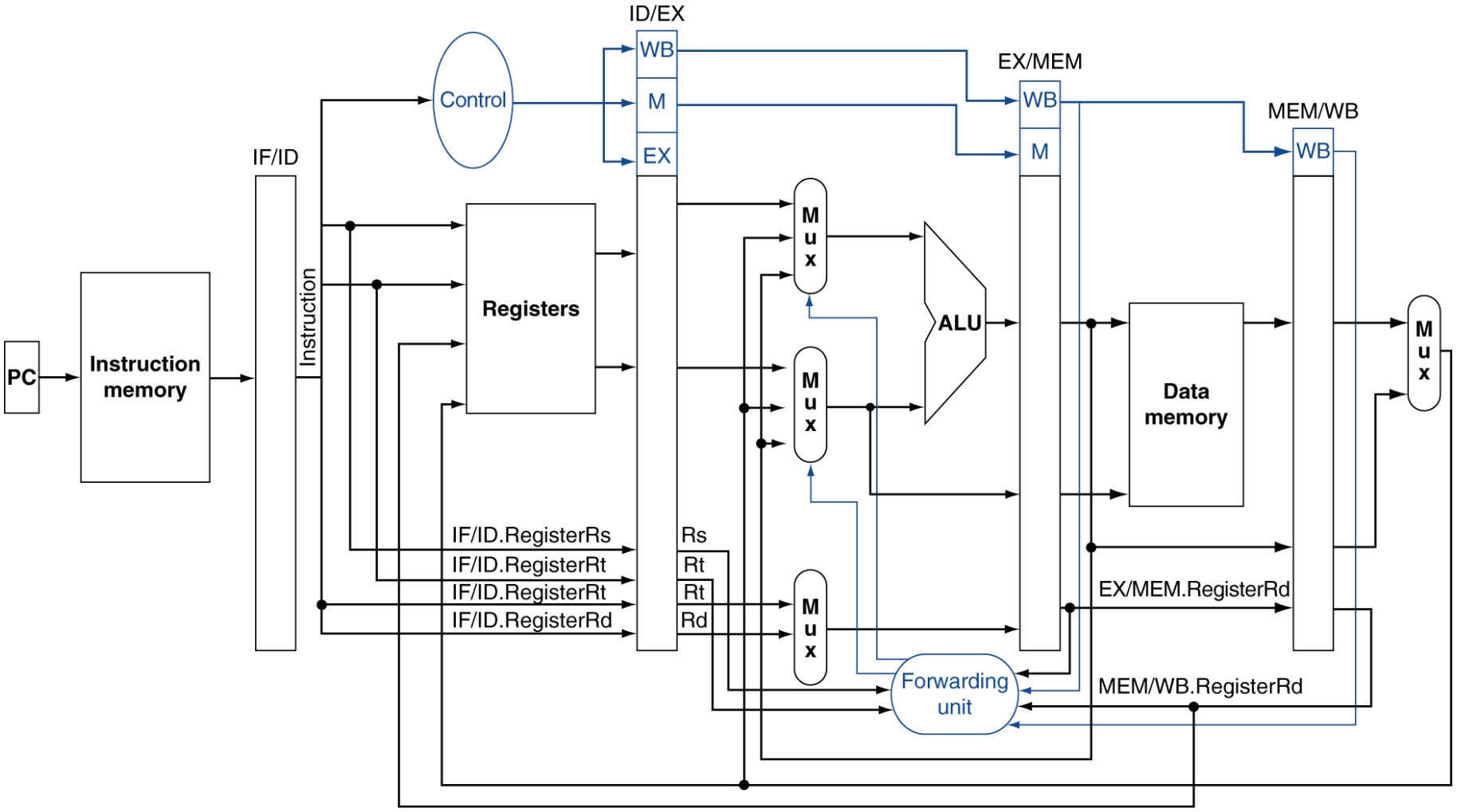
- Ambos os hazard ocorrem
  - Queremos usar o valor mais recente
- Revisar condições MEM hazard
  - Fwd apenas se condição EX hazard não for verdadeira

Hazards de dados: forwarding vs stalls



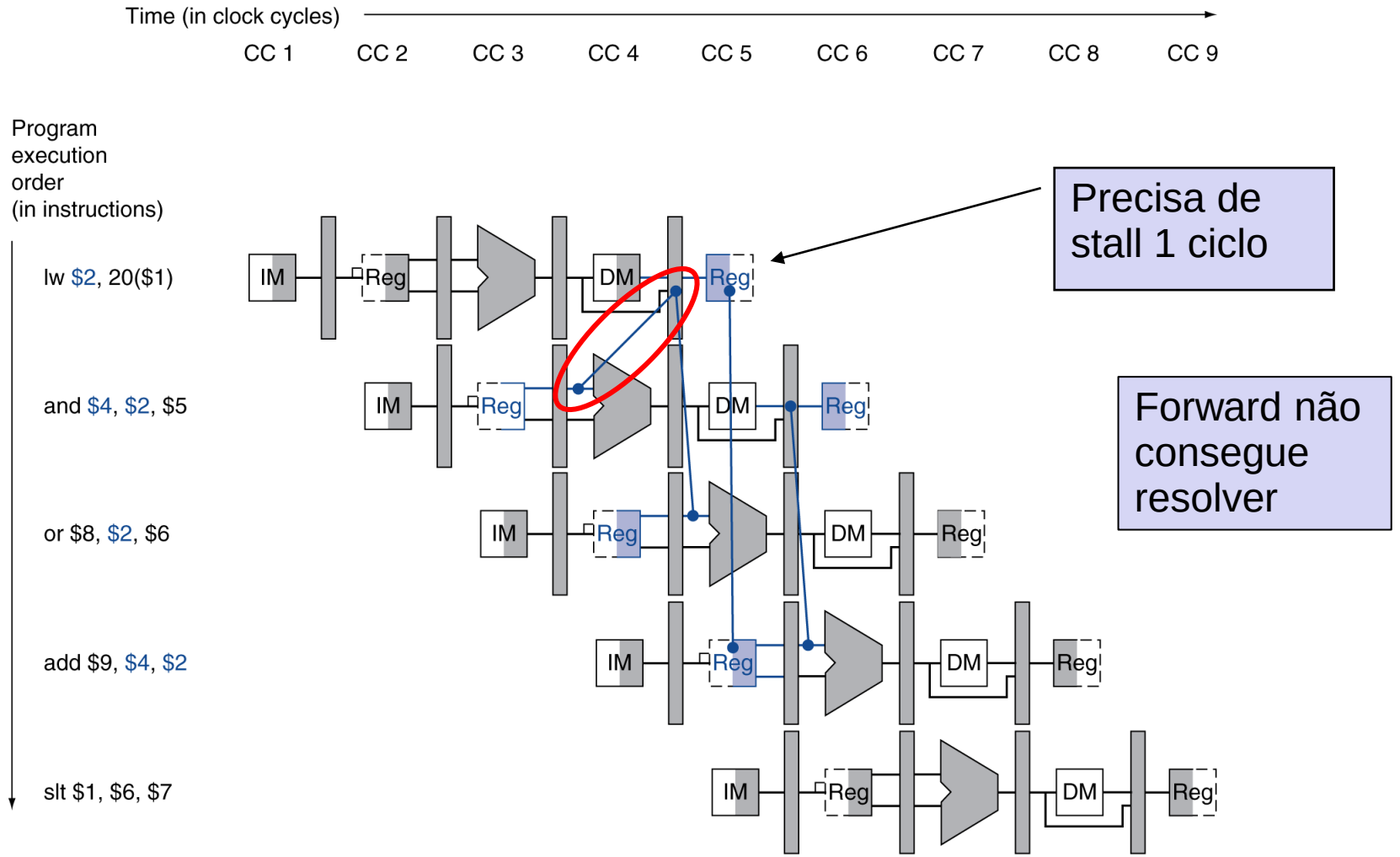
- MEM hazard
  - if (MEM/WB.RegWrite and (MEM/WB.RegisterRd ≠ 0)
    and not (EX/MEM.RegWrite and (EX/MEM.RegisterRd ≠ 0)
    and (EX/MEM.RegisterRd = ID/EX.RegisterRs))
    and (MEM/WB.RegisterRd = ID/EX.RegisterRs))
    ForwardA = 01
  - if (MEM/WB.RegWrite and (MEM/WB.RegisterRd ≠ 0)
    and not (EX/MEM.RegWrite and (EX/MEM.RegisterRd ≠ 0)
    and (EX/MEM.RegisterRd = ID/EX.RegisterRt))
    and (MEM/WB.RegisterRd = ID/EX.RegisterRt))
    ForwardB = 01

Hazards de dados: forwarding vs stalls

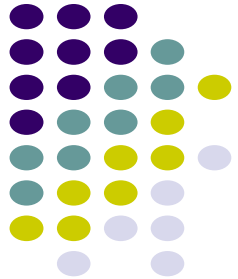


Hazards de dados: forwarding vs stalls

- Hazard de dados em LOAD

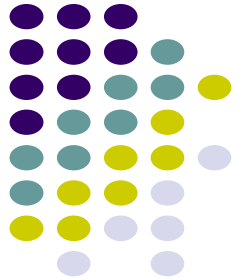


Hazards de dados: forwarding vs stalls



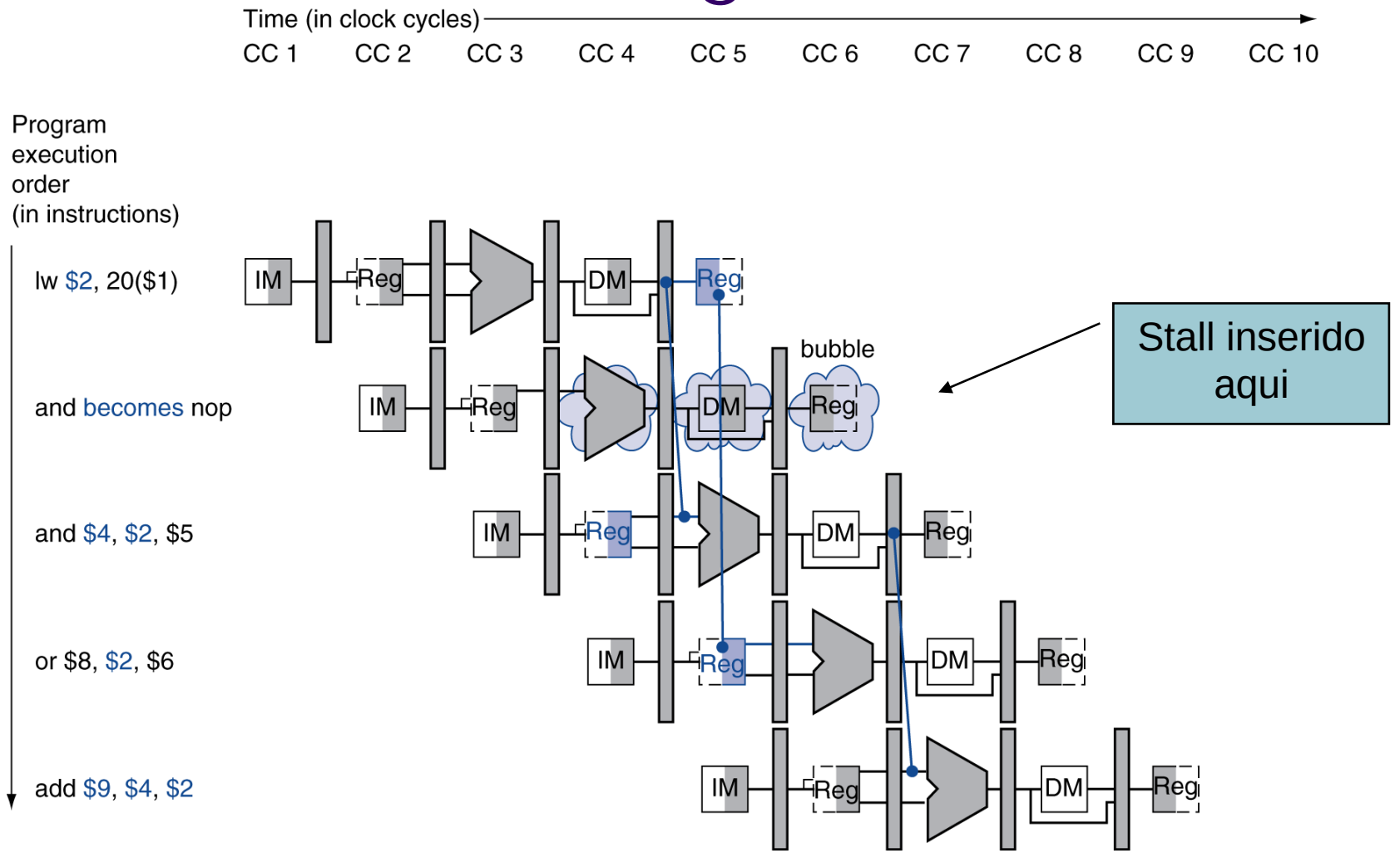
- Checar quando está usando no estágio ID
- Operandos da ALU são dados no ID
  - IF/ID.RegisterRs, IF/ID.RegisterRt
- Load-use hazard quando
  - ID/EX.MemRead and
    ((ID/EX.RegisterRt = IF/ID.RegisterRs) or
    (ID/EX.RegisterRt = IF/ID.RegisterRt))
- Se detectado, stall e insere bolha

Hazards de dados: forwarding vs stalls

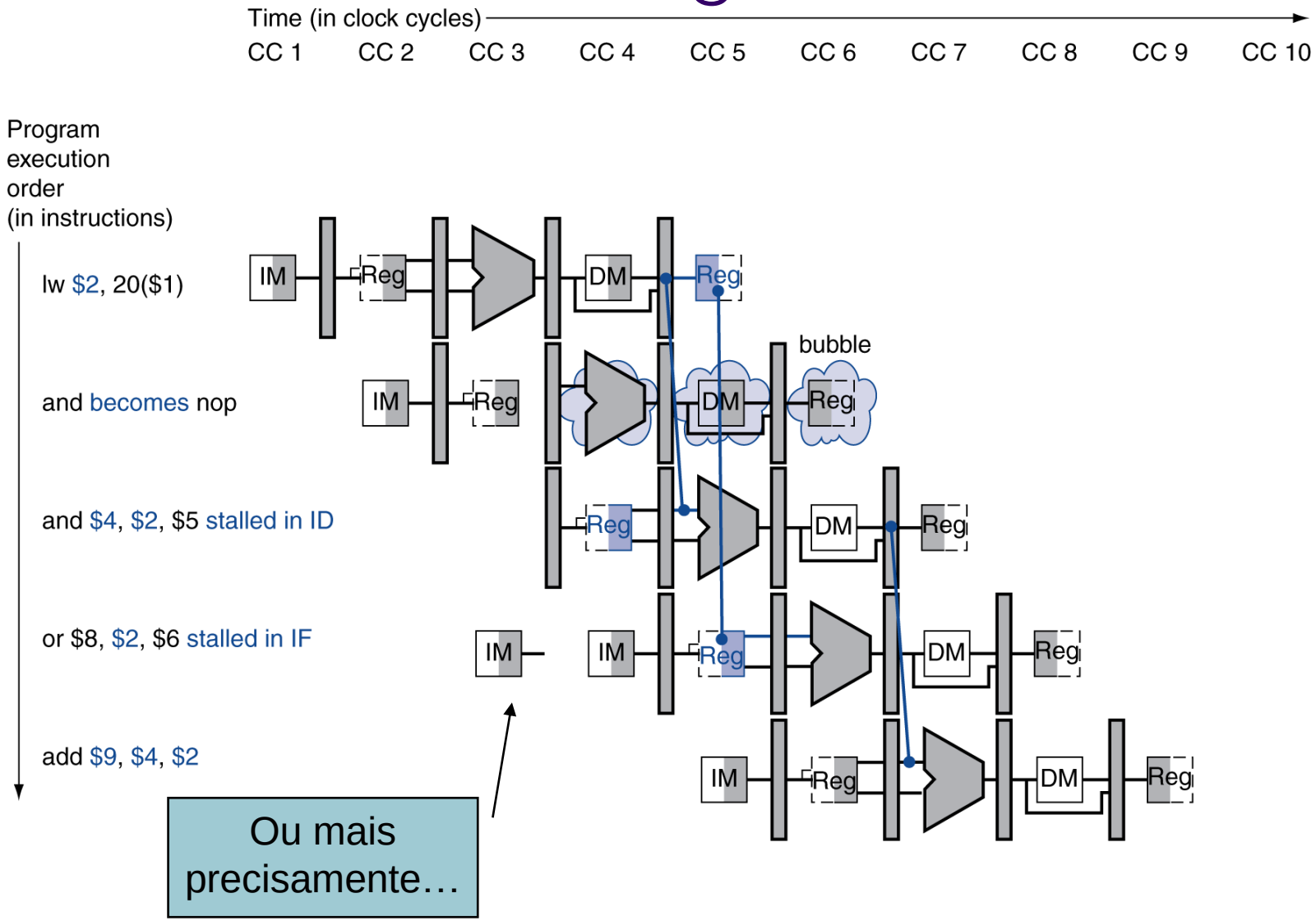


- Como fazer stall no pipeline?
  - Força os valores de controle no ID/EX para zero
    - EX, MEM and WB fazem **nop** (no-operation)
  - Impedir a atualização de PC e registrador IF/ID
    - Usar instrução decodificada novamente
    - A instrução seguinte é buscada novamente
    - 1 ciclo de stall permite MEM ler o dado de lw
      - Posteriormente pode encaminar para o estágio EX

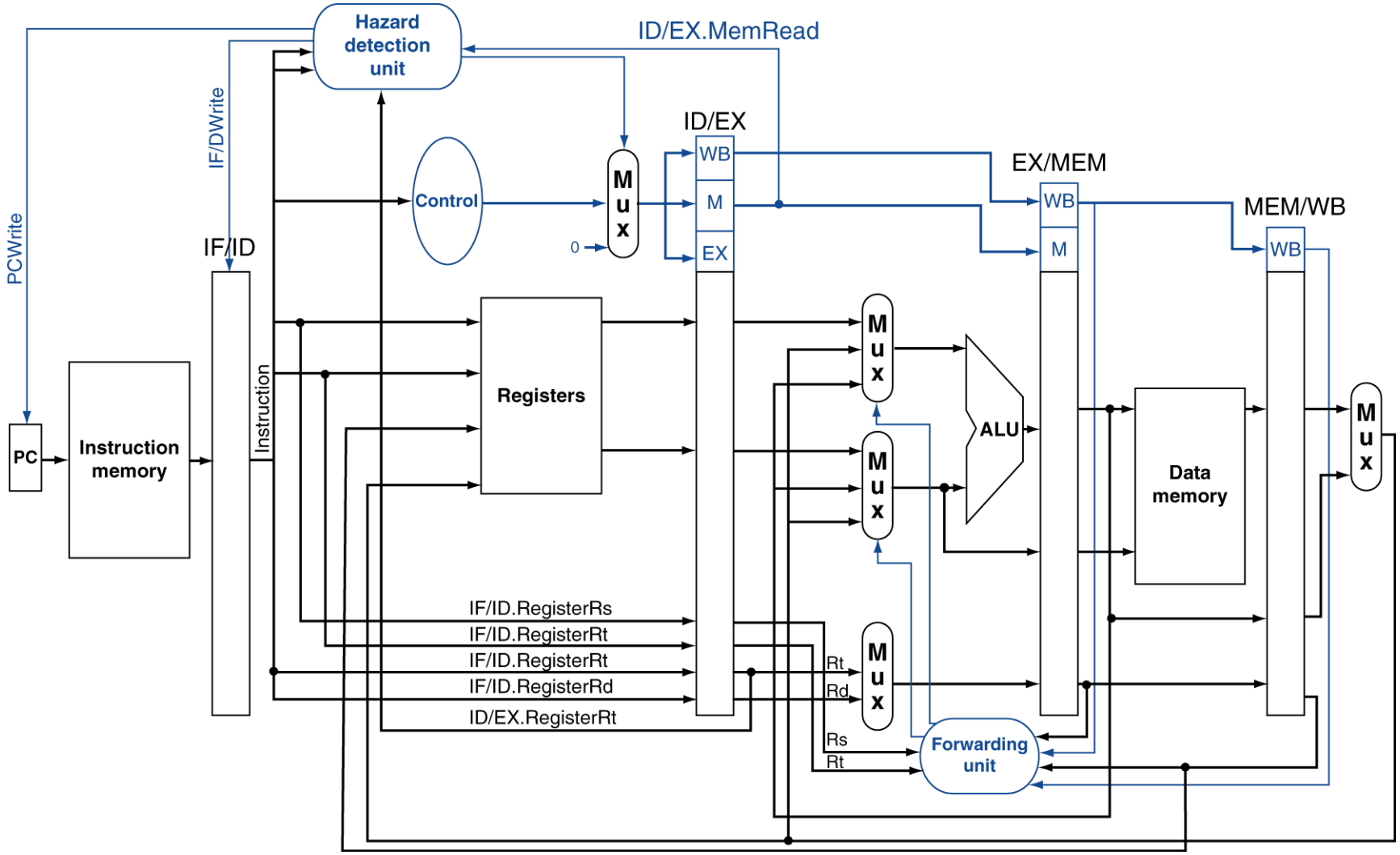
Hazards de dados: forwarding vs stalls



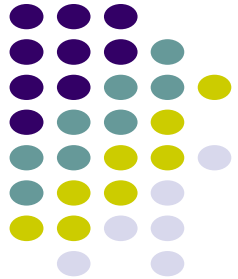
Hazards de dados: forwarding vs stalls



Caminho de dado com detecção de hazard



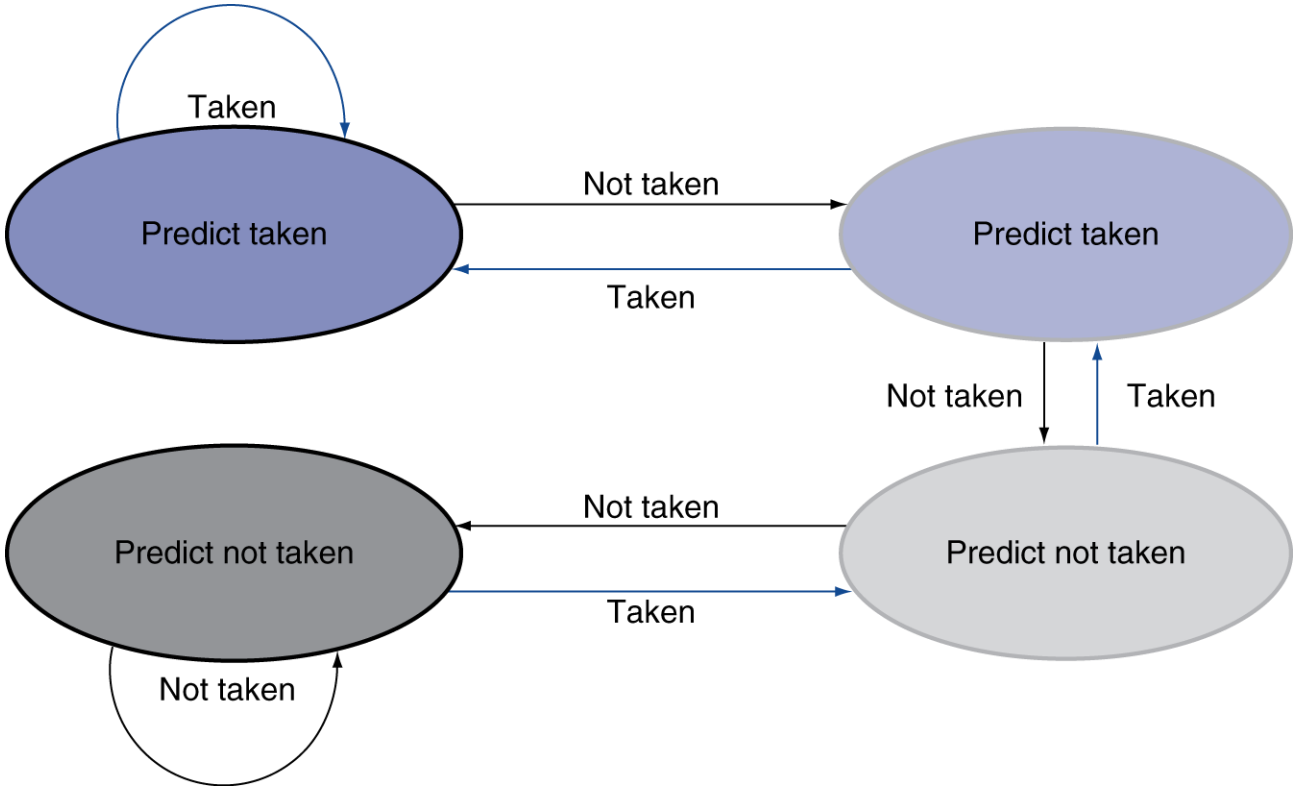
Hazards de dados: forwarding vs stalls



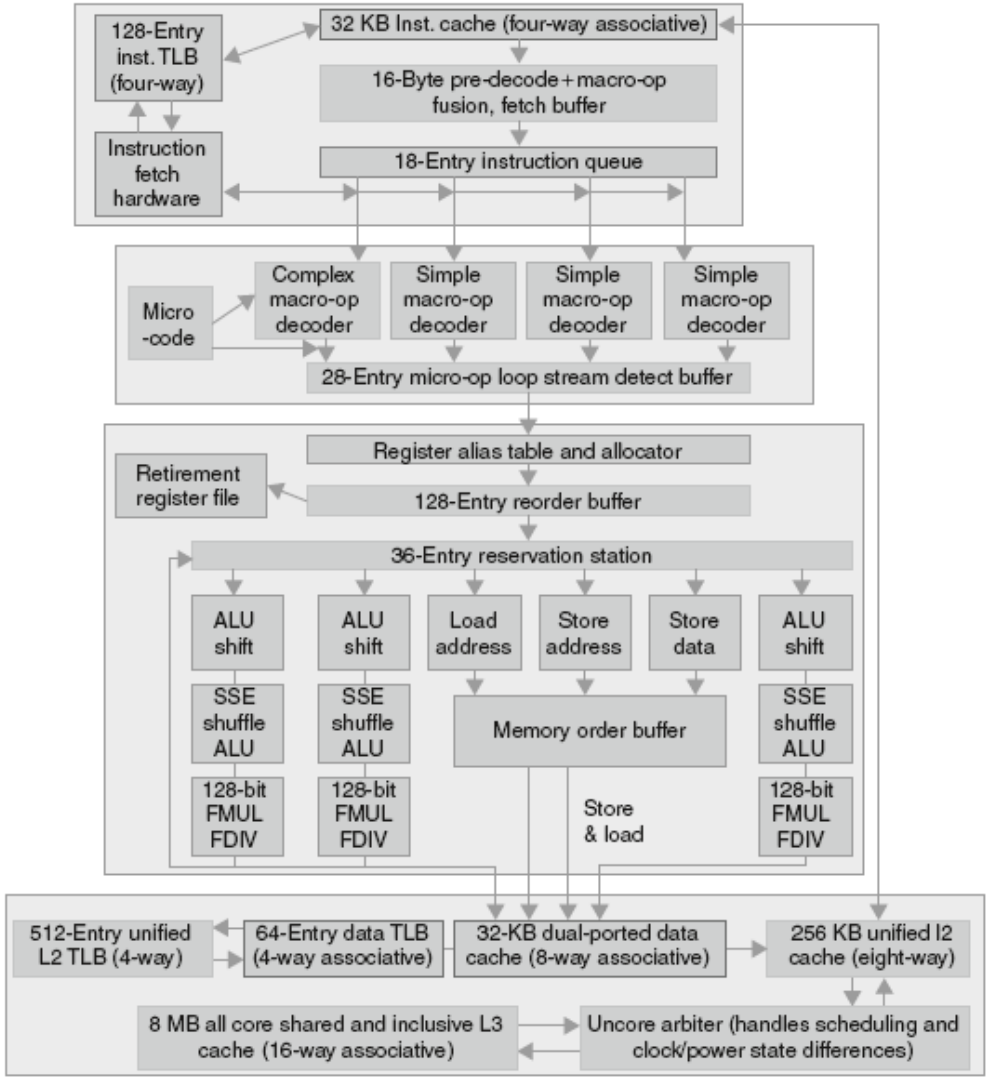
- Exemplo
  - Simulador Hades (Fig. 4.60)
    - Ex 1: Sem hazard
    - Ex 2: Hazard EXE
    - Ex 3: Hazard MEM
    - Ex 4: Exemplo LW

Preditor de salto de 2 bits

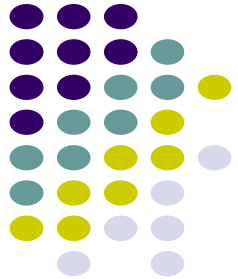
- Muda predição apenas se houver 2 falhas de predição subequentes



Pipeline do Core i7

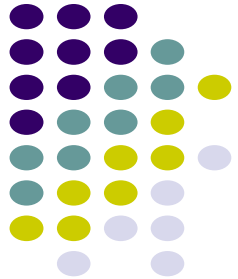


Verifique você mesmo



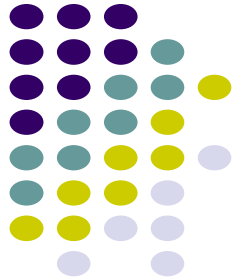
- Um grupo de alunos discutia sobre a eficiência de um pipeline de 5 estágios quando um deles apontou que nem todas as instruções estão ativas em cada estágio do pipeline. Depois de decidir ignorar os efeitos dos harzards, eles fizeram as cinco afirmações a seguir. Quais delas estão corretas?
  1. Permitir que jumps, branches e instruções da ALU utilizem menos estágios do que os 5 necessários pela instrução load aumentará o desempenho do pipeline sob todas as circunstâncias

Verifique você mesmo



2. Tentar permitir que algumas instruções utilizem menos ciclos não-ajuda, pois a vazão é determinada pelo ciclo de clock; o número de estágios do pipe por instrução afeta a latência, e não a vazão
3. Permitir que jumps, branches e instruções da ALU utilizem menos ciclos só ajuda quando nem loads nem stores estão no pipeline, de modo que os benefícios são pequenos
4. Você não pode fazer com que as instruções da ALU utilizem menos ciclos, devido à escrita do resultado, mas os branches e jumps podem utilizar menos ciclos, de modo que existe alguma oportunidade de melhoria

Verifique você mesmo



5. Em vez de tentar fazer com que as instruções utilizem menos ciclos de clock, devemos explorar um meio de tornar o pipeline mais longo, de modo que as instruções utilizem mais ciclos, porém com ciclos mais curtos. Isso poderia melhorar o desempenho

Verifique você mesmo

- Resposta: afirmações 2 e 5 estão corretas

Referências

- PATTERSON, D. A. ; HENNESSY, J.L. Organização e projeto de computadores – a interface hardware software. 3. ed. Editora Campus, 2005.
- STALLINGS, W. Arquitetura e organização de computadores: projeto para o desempenho. 8. ed. Prentice Hall, 2009.