Arquitetura de Computadores I

Pipeline

-- Conflito de dados – paradas e adiantamentos –

-- Conflito de controle – detecção de desvios e descarte de instruções --

Edson Moreno

edson.moreno@pucrs.br

http://www.inf.pucrs.br/~emoreno

Conflito de dados

Parada / Bolha / Stall

Organização do MIPS: pipeline

- Conflito de dados e parada

Nem sempre o adiantamento irá resolver um conflito de dados.

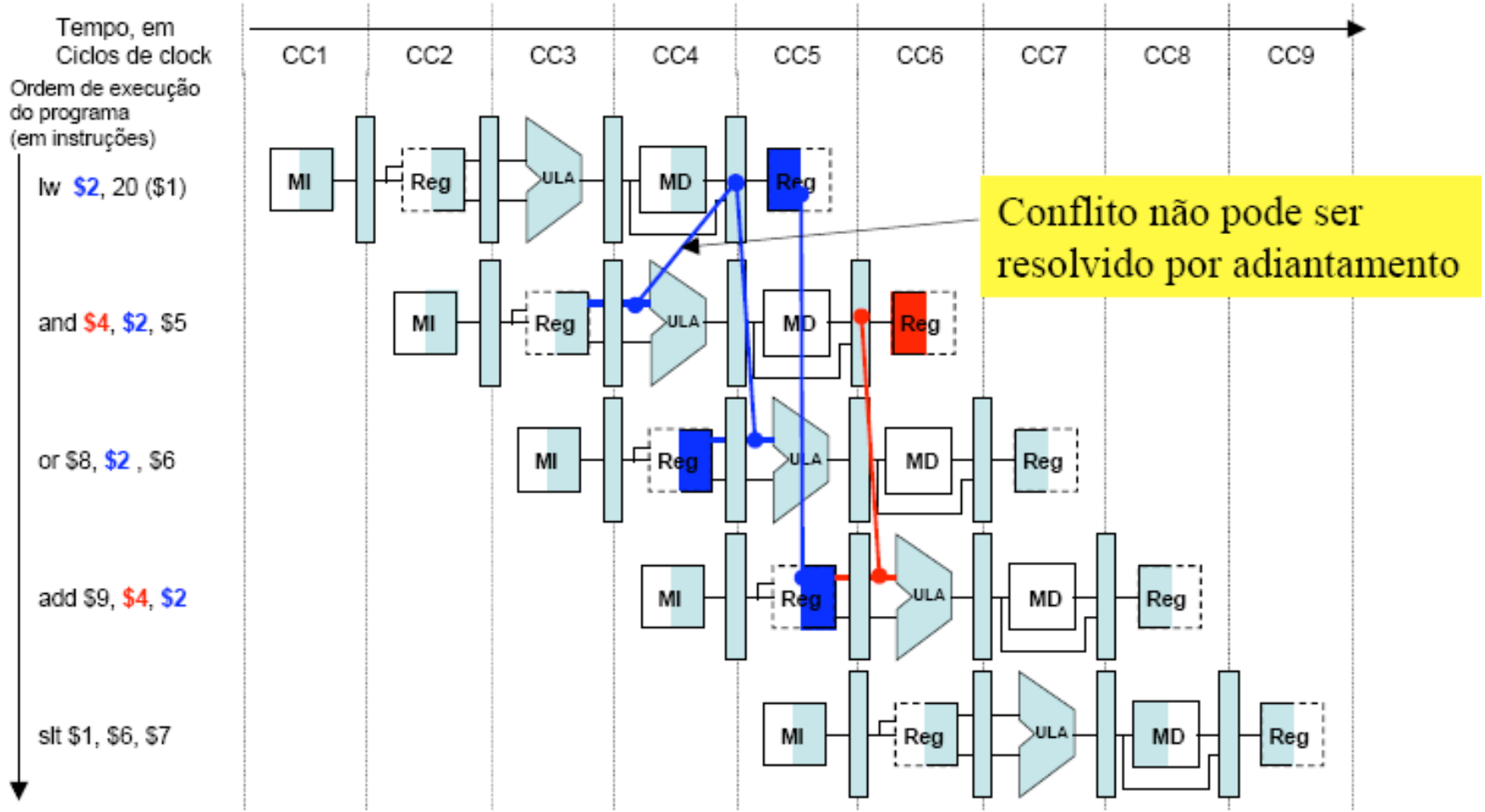
Exemplo:

```
lw    $2, 20($1) # registrador $2 é escrito
and   $4, $2, $5 # primeiro operando ($2) depende de lw; registrador $4 é escrito
or    $8, $2, $6 # primeiro operando ($2) depende de lw
add   $9, $4, $2 # prim. operando ($4) depende de and e seg.($2) de lw
slt   $1, $6, $7 # nenhuma dependência
```

...

Organização do MIPS: pipeline

- Conflito de dados e parada



Organização do MIPS: pipeline

- Conflito de dados e parada

  Unidades de detecção de conflitos

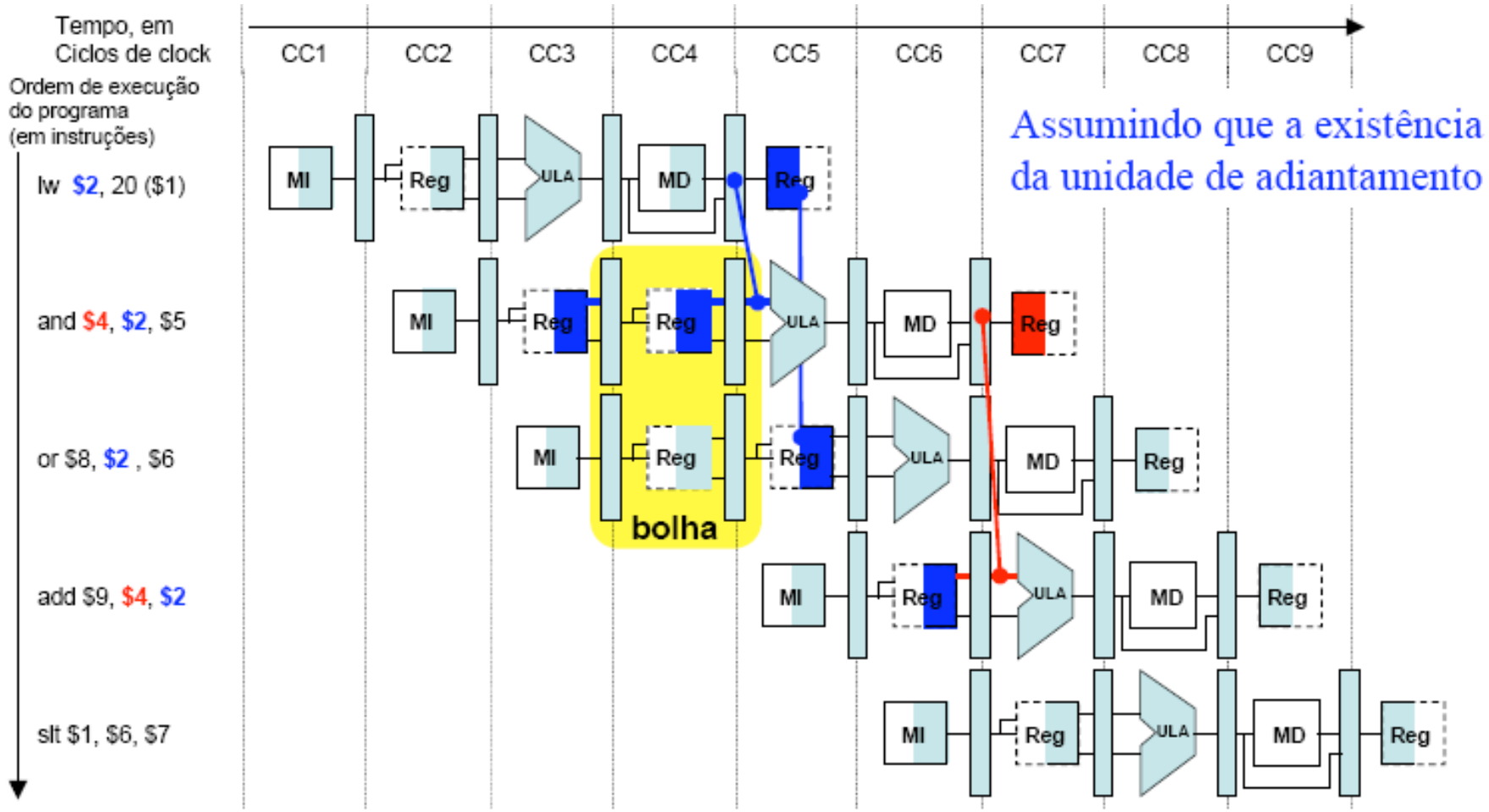
  - Faz o pipeline parar quando houver uma instrução load word, seguida da instrução que leia o registrador onde esta instrução de load word escreveu
  - Vai operar durante o estágio DI, inserindo uma parada entre a instrução load word e o uso de seu resultado
  - Condição a ser verificada:

```
Se (
    (DI/EX.LerMem = 1) e
    (
      (DI/EX.RegistradorRt = BI/DI.RegistradorRs) ou
      (DI/EX.RegistradorRt = BI/DI.RegistradorRt)
    )
) então « Pára o pipeline por um ciclo de relógio »
```

Load é a única instrução que lê dados da memória

Organização do MIPS: pipeline

- Conflito de dados e parada



Organização do MIPS: pipeline

- Conflito de dados e parada

  Trancando o prosseguimento das instruções posteriores a uma instrução de load word

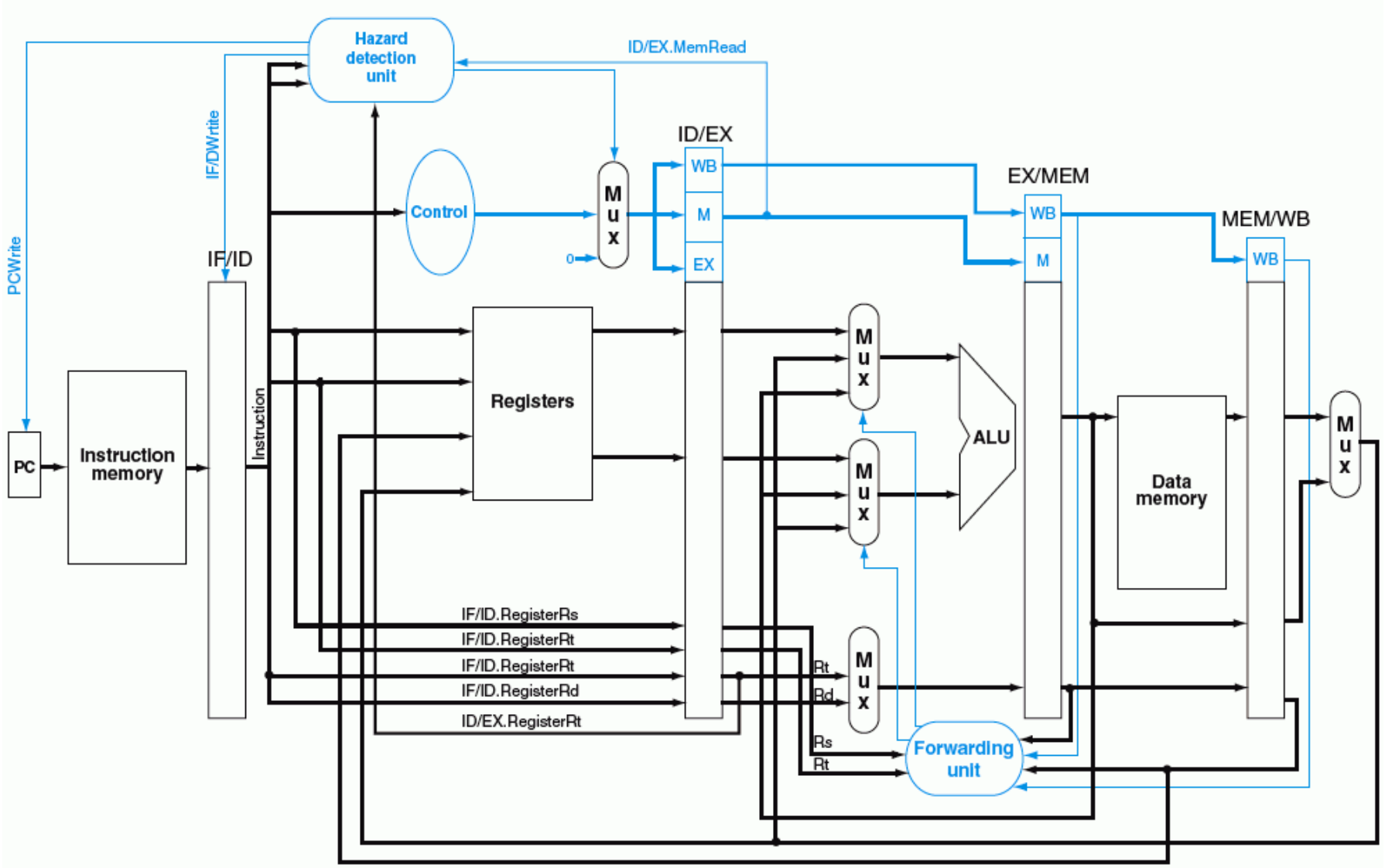
  - Se a instrução que está no estágio DI estiver parada, então o estágio BI também precisa parar
  - Para impedir o avanço das instruções pelo pipeline, basta evitar que tanto o PC quanto o registrador BI/DI sejam escritos
  - As condições do item anterior fazem com que, no ciclo de relógio seguinte:
    - A instrução que está em BI seja lida novamente
    - Os registradores lidos em DI serão lidos novamente

Organização do MIPS: pipeline

- Propagando uma bolha pelo pipeline
  - Como a instrução load word prossegue pelo pipeline, cria-se uma «bolha» de execução, a qual deve também prosseguir pelo pipeline
  - Uma «bolha» deve executar em cada estágio o mesmo que uma instrução NOP executa

  - NOP
    - Todos os sinais de controle em 0 (zero) para os estágios EX, MEM, ER
    - Estes valores de sinal de controle são passados adiante a cada ciclo de relógio, produzindo o efeito desejado (nenhum registrador ou memória é alterado/escrito)

Organização do MIPS: pipeline



Conflito de dados

Adiantamento / Fowarding / Bypass

Organização do MIPS: pipeline

- Conflito de dados

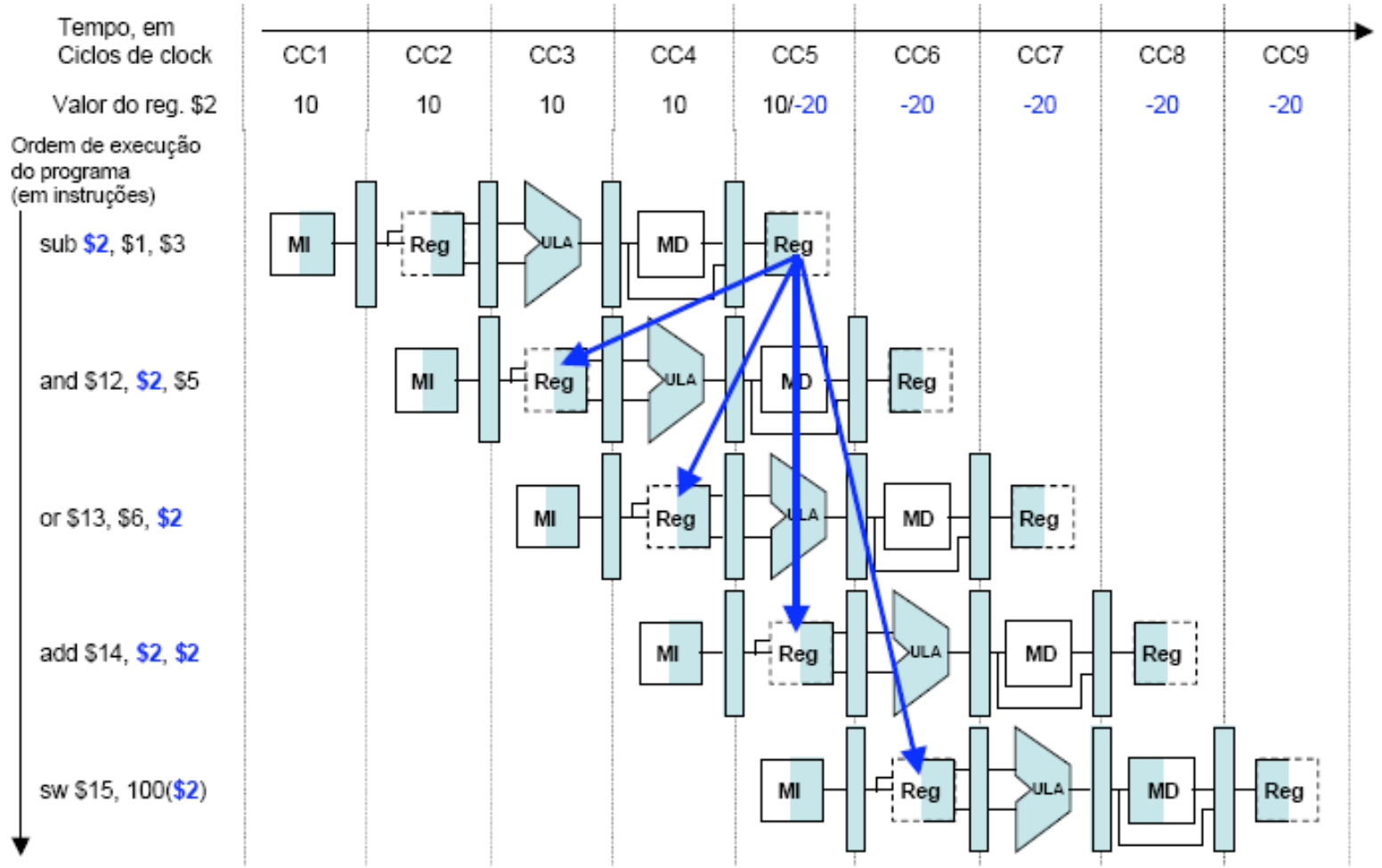
  Seja o seguinte trecho de código, escrito para o MIPS

```
sub   $2, $1, $3          # registrador $é é escrito pela instrução sub
and   $12, $2, $5  # primeiro operando ($2) depende de sub
or    $13, $6, $2  # segundo operando ($2) depende de sub
add   $14, $2, $2  # prim. e seg. operandos ($2) dependem de sub
sw    $15, 100($2)          # base ($2) depende de sub
```

  Para estudar as conseqüências destas dependências quando da execução em pipeline, usar um diagrama de pipeline de múltiplos ciclos.

Organização do MIPS: pipeline

- Conflito de dados



Organização do MIPS: pipeline

- Conflito de dados

  Uma solução seria o compilador evitar seqüências de instruções que gerem conflitos de dados

```
sub   $2, $1, $3              # registrador $2 é escrito pela instrução sub
nop                           # na falta de instruções de sejam independentes, o
nop                           # compilador inseriria instruções «nop»
and   $12, $2, $5  # primeiro operando ($2) depende de sub
or    $13, $6, $2  # segundo operando ($2) depende de sub
add   $14, $2, $2  # prim. e seg. operandos ($2) dependem de sub
sw    $15, 100($2)            # base ($2) depende de sub
```

  Problemas?

  Conflitos de dados são muito freqüentes

  A inserção de instruções nop causam perda de desempenho

Organização do MIPS: pipeline

- Conflito de dados

  Outra solução

  - Detectar o conflito
  - Adiantar o resultado da ULA (ou da memória de dados)

  Testes para detecção de conflitos (uso de registardores de pipeline)

  1a. DI/EX.RegistradorRs = EX/MEM.RegistradorRd

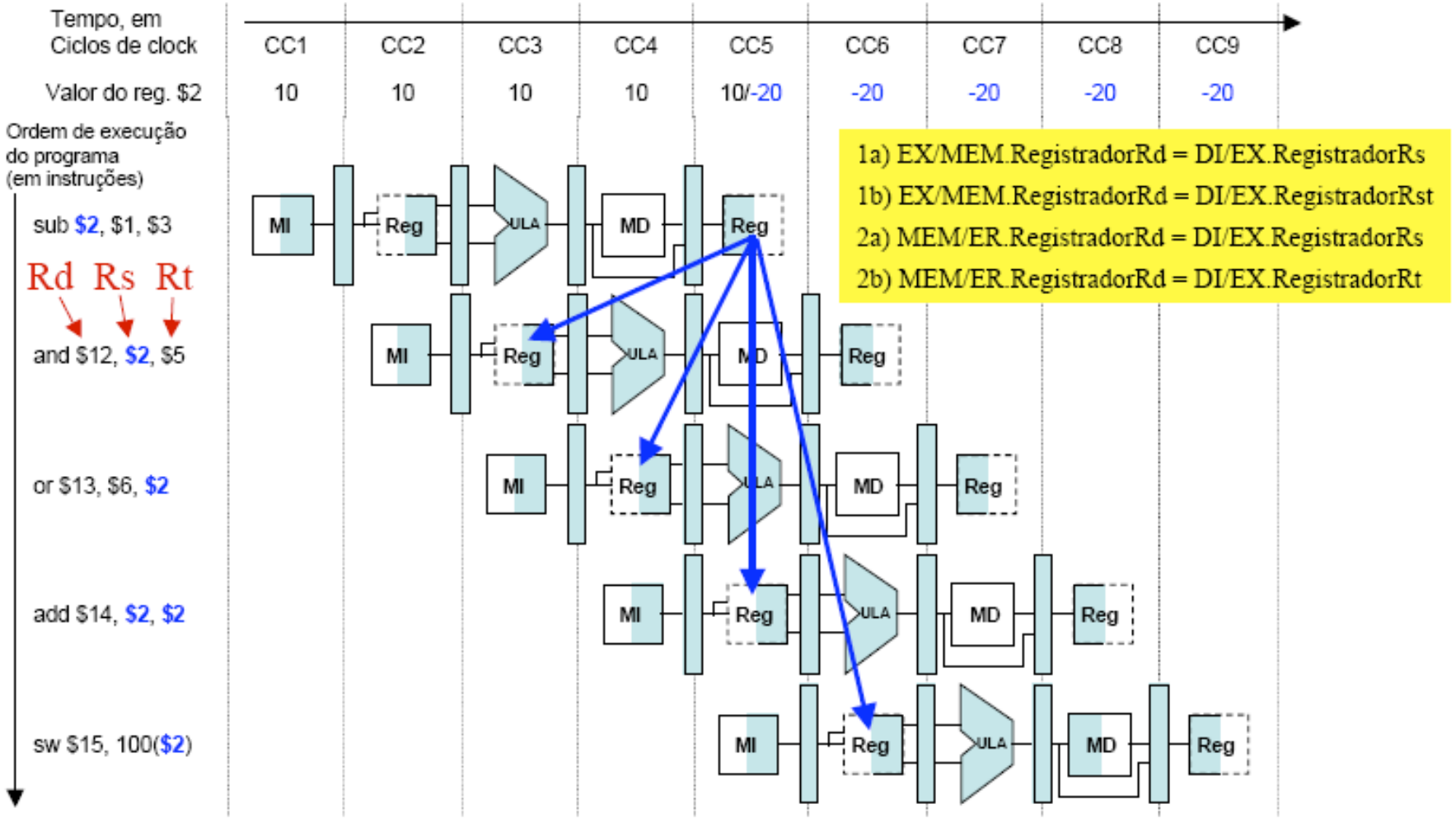
  1b. DI/EX.RegistradorRt = EX/MEM.RegistradorRd

  2a. DI/EX.RegistradorRs = MEM/ER.RegistradorRd

  2b. DI/EX.RegistradorRt = MEM/ER.RegistradorRd

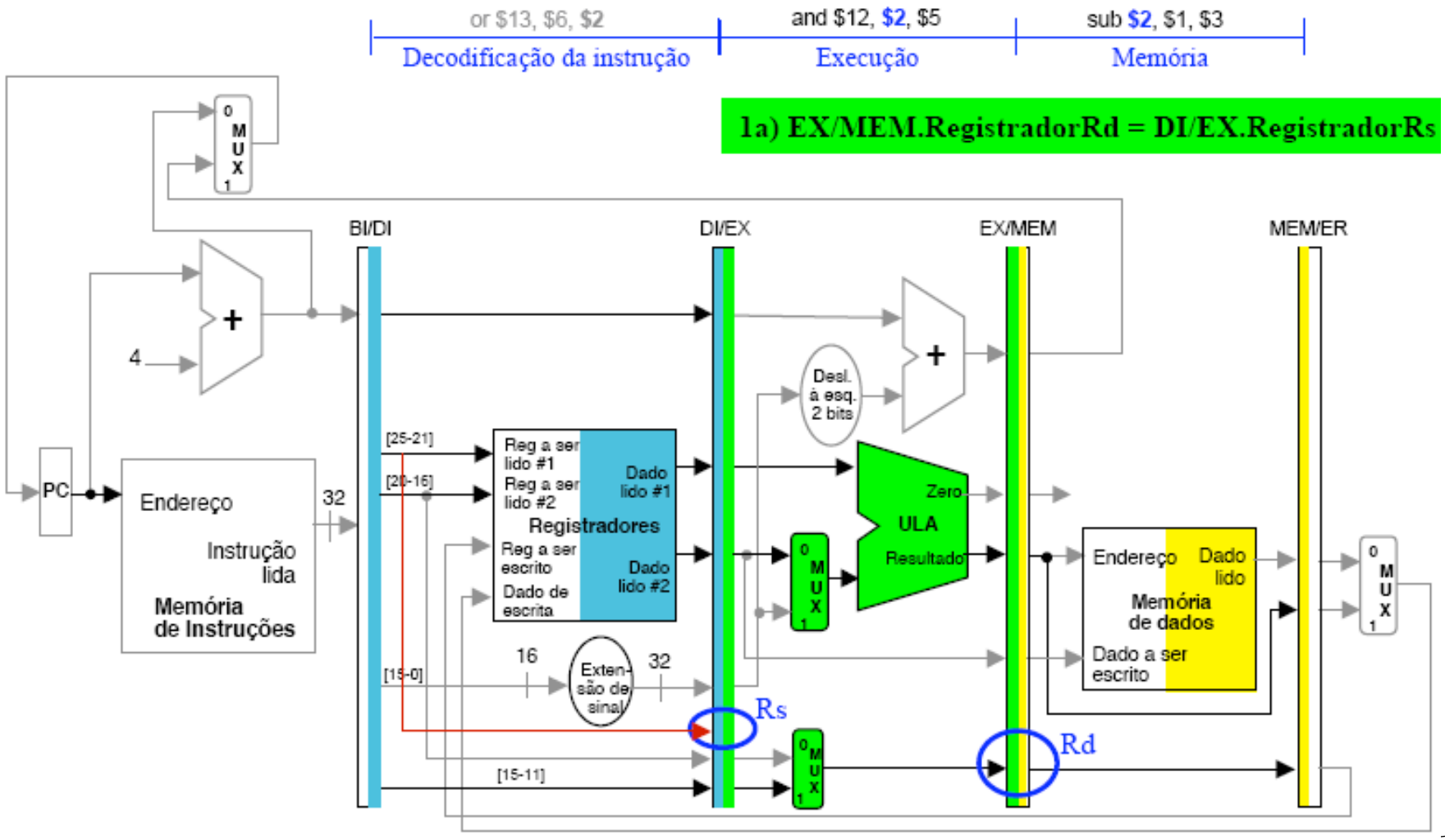
Organização do MIPS: pipeline

- Conflito de dados



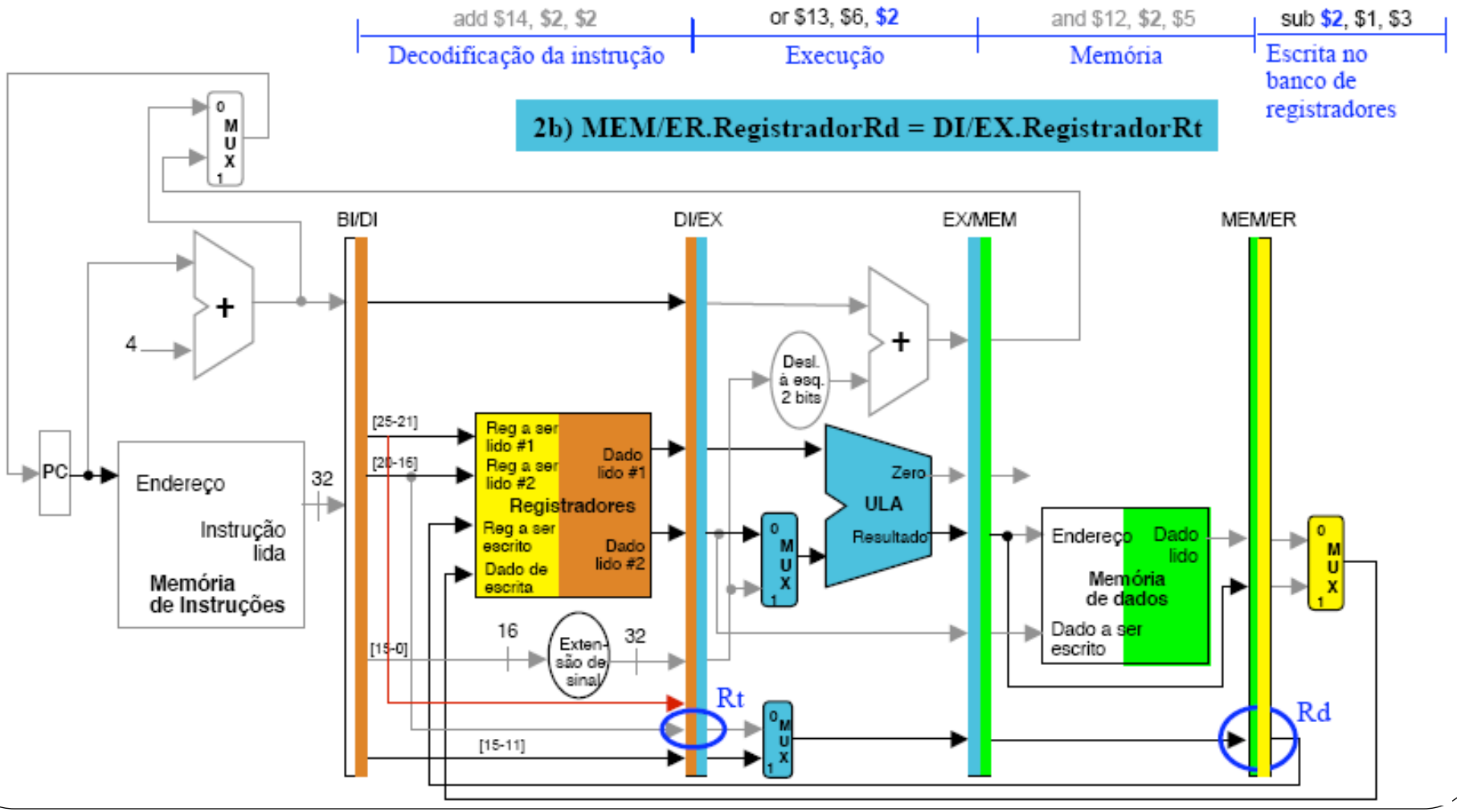
Organização do MIPS: pipeline

- Conflito de dados



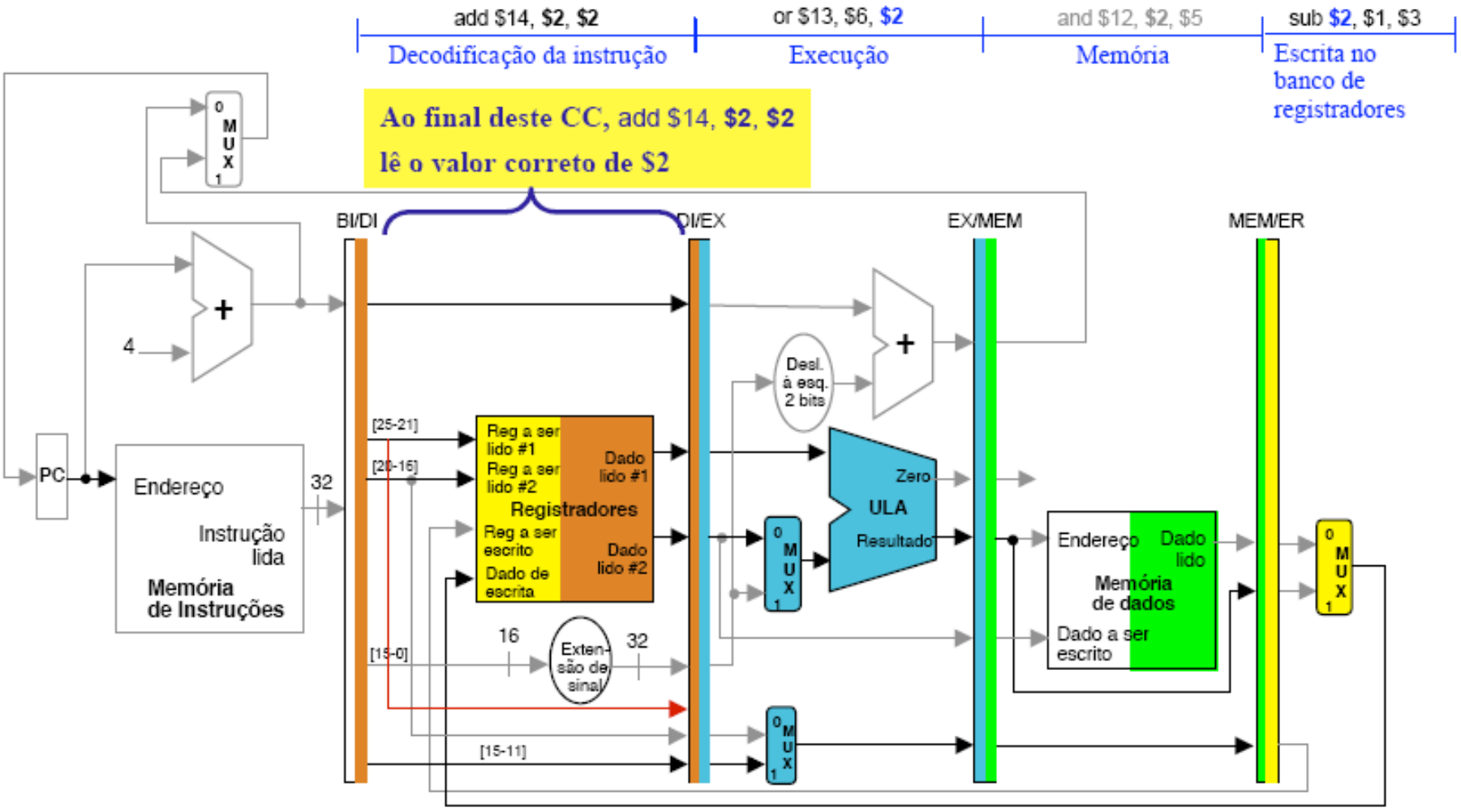
Organização do MIPS: pipeline

- Conflito de dados



Organização do MIPS: pipeline

- Conflito de dados



Organização do MIPS: pipeline

- Detectando e resolvendo conflitos de dados

1. Conflito no estágio EX

Se (
    (EX/MEM.EscReg = 1) e
    (EX/MEM.RegistradorRd ≠ 0) e
    (EX/MEM.RegistradorRd = DI/EX.RegistradorRs)
) Então Adianta.opA = **10**

Se (
    (EX/MEM.EscReg = 1) e
    (EX/MEM.RegistradorRd ≠ 0) e
    (EX/MEM.RegistradorRd = DI/EX.RegistradorRt)
) Então Adianta.opB = **10**

Organização do MIPS: pipeline

- Detectando e resolvendo conflitos de dados

2. Conflito no estágio MEM

Se (

(MEM/ER.EscReg = 1) e

(MEM/ER.RegistradorRd ≠ 0) e

(MEM/ER.RegistradorRd = DI/EX.RegistradorRs)

) Então Adianta.opA = **01**

Se (

(MEM/ER.EscReg = 1) e

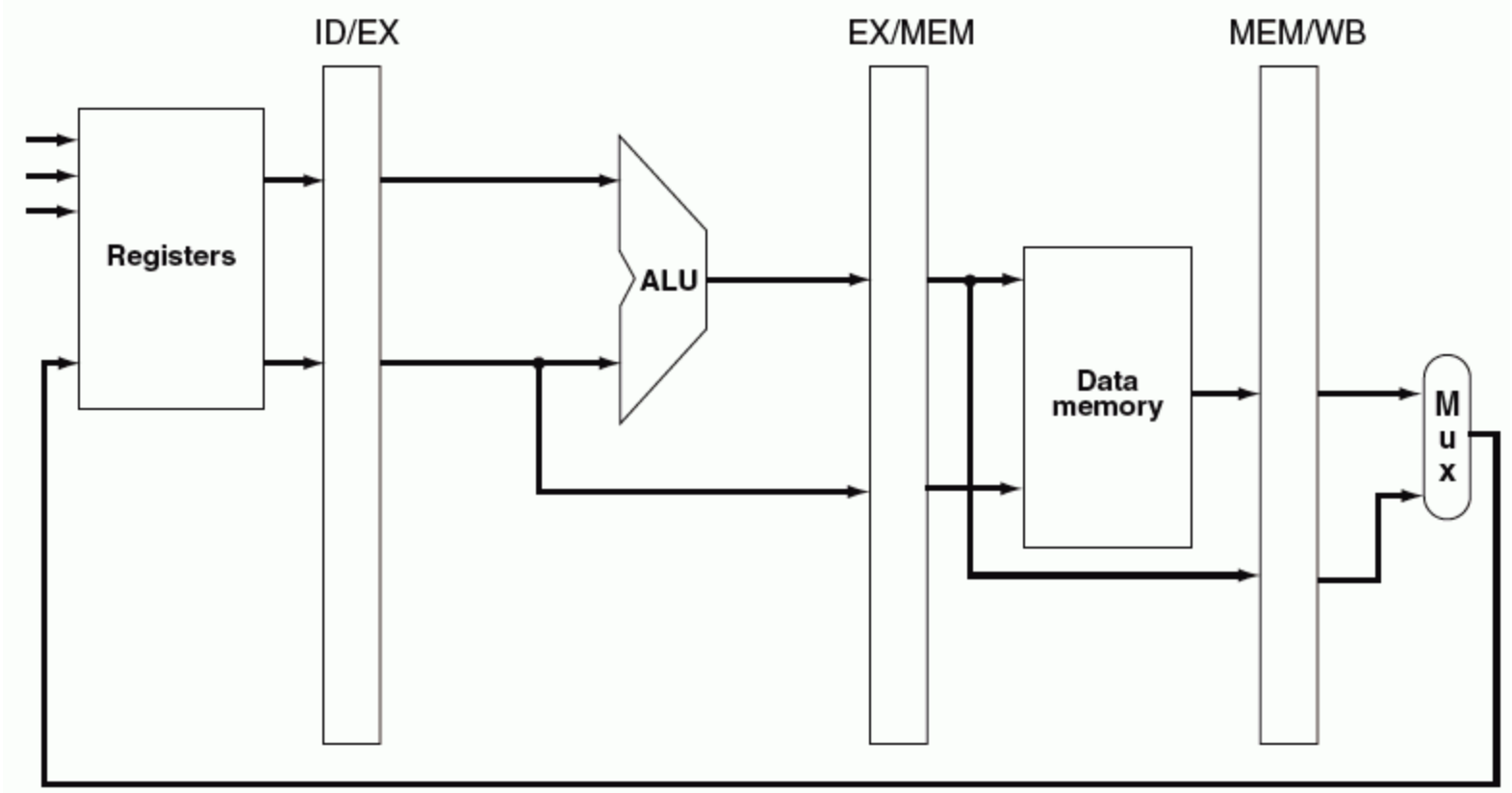
(MEM/ER.RegistradorRd ≠ 0) e

(MEM/ER.RegistradorRd = DI/EX.RegistradorRt)

) Então Adianta.opB = **01**

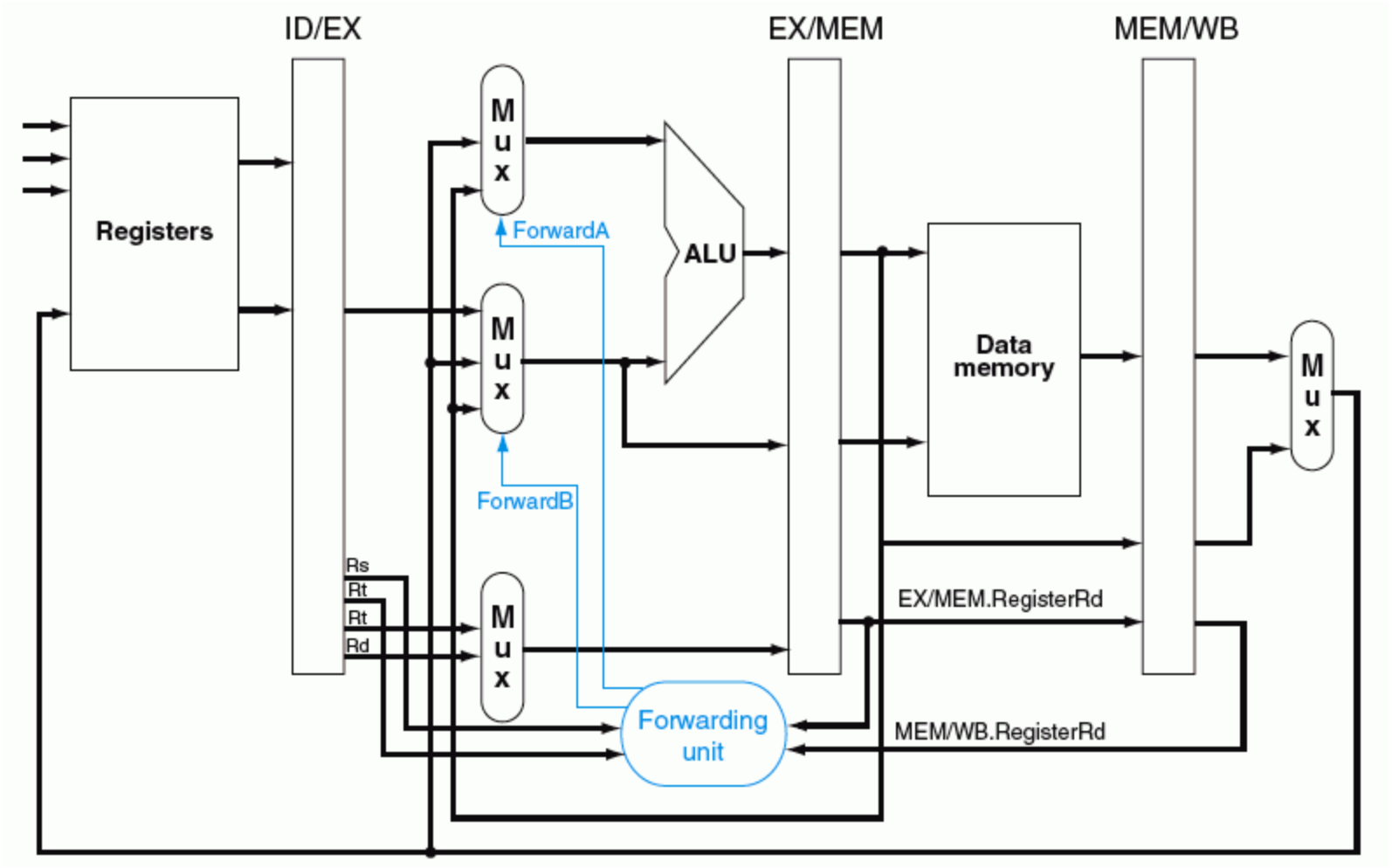
Organização do MIPS: pipeline

- Arquitetura sem adiantamento



Organização do MIPS: pipeline

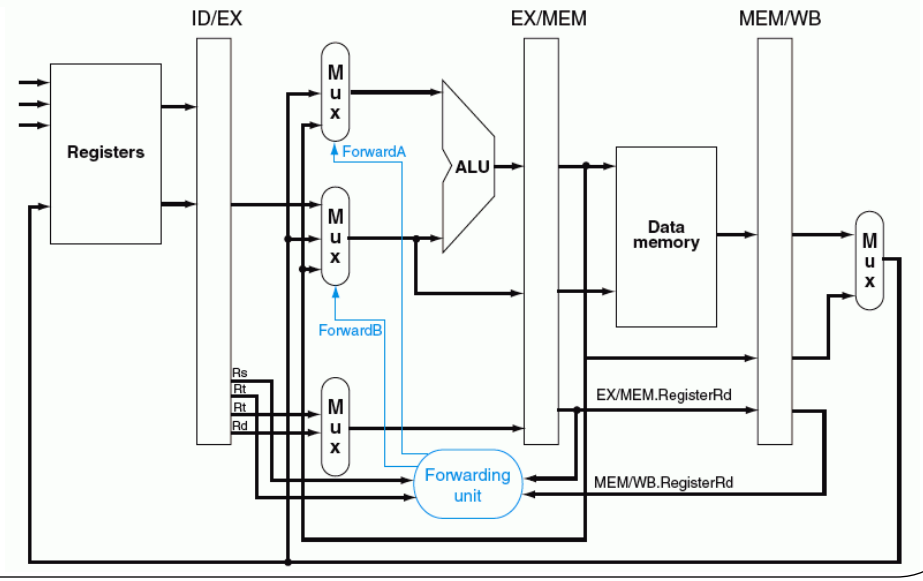
- Arquitetura com adiantamento



Organização do MIPS: pipeline

- Arquitetura com adiantamento

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.



Organização do MIPS: pipeline

- Complicação
  - Conflito entre o resultado do estágio ER e o resultado no estágio MEM e o operando-fonte da instrução no estágio da ULA.

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

Organização do MIPS: pipeline

- Detectando e resolvendo conflitos de dados
  2. Conflitos no estágio MEM

se(

(MEM/ER.EscReg = 1) e

(MEM/ER.RegistradorRd ≠ 0) e

(EX/MEM.RegistradorRd ≠ DI/EX.RegistradorRs) e

(MEM/ER.RegistradorRd = DI/EX.RegistradorRs)

) então Adianta.opA = 01

se(

(MEM/ER.EscReg = 1) e

(MEM/ER.RegistradorRd ≠ 0) e

(EX/MEM.RegistradorRd ≠ DI/EX.RegistradorRt) e

(MEM/ER.RegistradorRd = DI/EX.RegistradorRt)

) então Adianta.opB = 01

Conflito de controle

Tratamento de desvios e descarte de instruções

Detecção de desvios

- Ocorrência é menor do que os conflitos de dados
  - Mecanismos de adiantamento e parada tem maior emprego
- Mecanismos de tratamento
  - Decisão estática
  - Decisão dinâmica
- Instruções de desvio
  - Reconhecidas somente a partir do segundo estágio

Consideração estática

- Desvio não é tomado
  - Conforme visto, bolhas afetam o desempenho do pipeline
    - Boa solução é assumir que o salto nunca ocorre
  - Mas e se o salto ocorrer?
    - Condição somente é validada no final do terceiro estágio
    - Instruções presentes nos estágios BI, DI e EX tem de ser eliminadas
      - Processo similar a inserção de bolhas quando da ocorrência de uma parada

Consideração estática

- Desvio não é tomado
  - Eliminando instruções inválidas do pipeline
    - Processo conhecido como *flush*
    - Elimina as instruções através do reset das barreiras de registradores BIDI, DIEX e EXME
      - Registradores de dado e sinais de controle representam instrução NOP

Consideração estática

- Como reduzir o atraso dos desvios
  - Reduzindo o custo do desvio tomado
    - Adiantando um estágio, reduz a penalidade
      - PC desviado somente válido no estágio ME, após EX
      - Menos instruções carregadas inutilmente
      - Melhora a vazão de instruções válidas
  - Maioria dos desvios contam com testes simples
    - Observação feita por projetistas
    - Consideram a igualdade ou sinal (qualificadores)
    - Não exigem operação com a ULA
      - Podem ser realizados com poucas portas lógicas

Consideração estática

- Como reduzir o custo do desvio tomado
  - Antecipação do desvio exige
    - Mecanismo para cálculo do novo PC
      - Presente desde a barreira BIDI
        - PC + 4 + Valor imediato
      - Somador destes valores pode ser alocado no estágio DI
        - Cálculo realizado para todas instruções
    - Mecanismo de avaliação do desvio
      - Deve considerar valores lidos no banco de registradores (beq)
      - Igualdade pode ser testada a partir de um xor bit a bit + or
        - Zero representa que todos bits são iguais
      - Mover tal procedimento para o estágio DI implica hw adicional
        - Deve prever conflito de dados (dependência verdadeira)

Consideração estática

- Complicadores da antecipação do teste de igualdade
  - Se realizado durante o estágio DI, exige
    - Decodificar a instrução
    - Decidir pelo uso ou não da unidade de igualdade
    - Completar a comparação de igualdade
      - Se desvio ocorrer, novo PC tem de ser liberado
  - Valores necessário para desvio podem sofrer com parada
    - Operando de comparação pode
      - Depender de uma operação que está sendo realizada na ULA
      - Depender de uma leitura na memória

Organização do MIPS: pipeline

- Conflito de controle

  Desvio condicional em pipeline

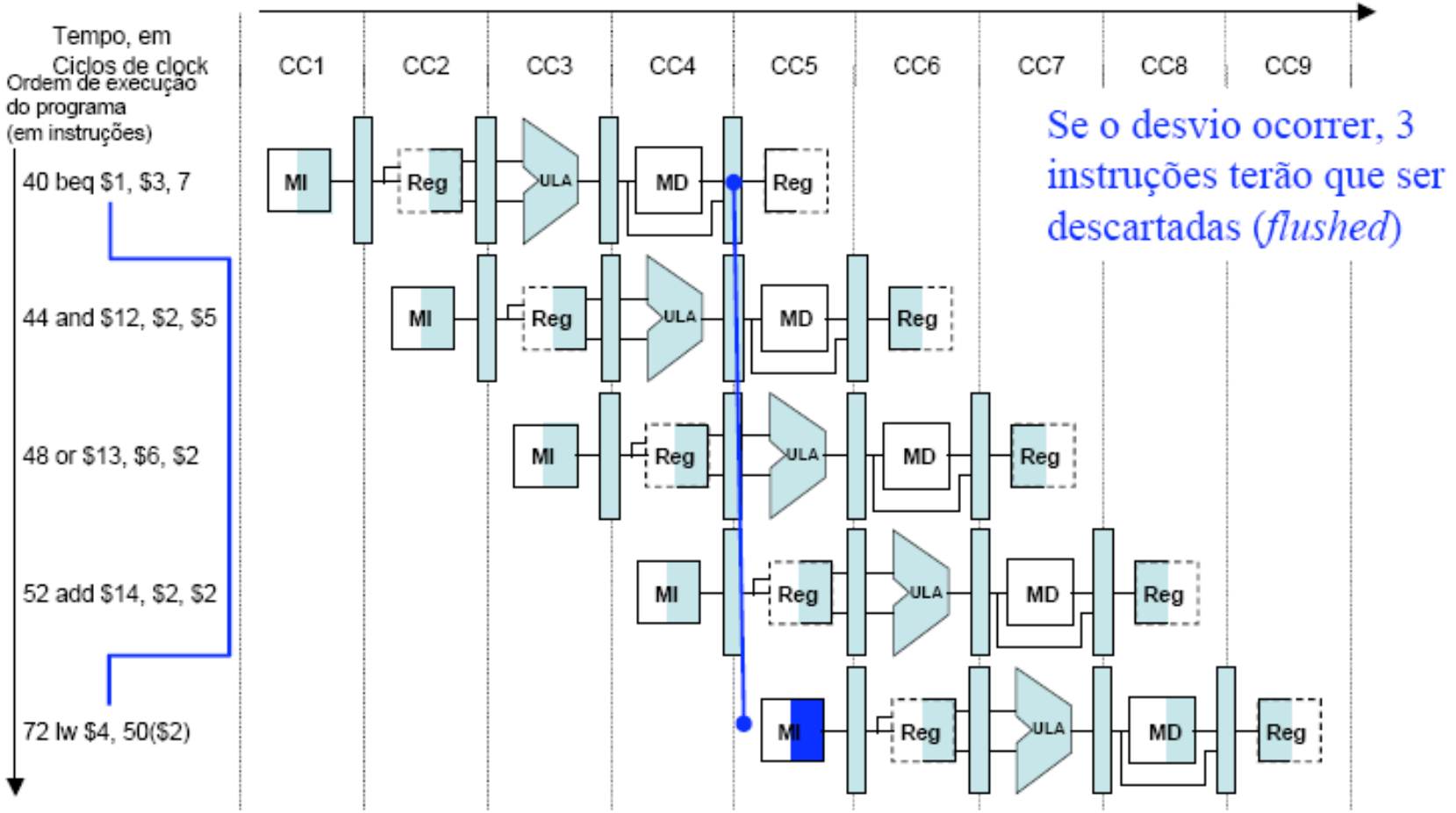
  Exemplo

```
36  sub     $10, $4, $8
40  beq     $1, $3, 7      # desvio relativo ao PC para 40 + 4 + (7*4) = 72
44  and     $12, $2, $5
48  or      $13, $2, $6
52  add     $14, $4, $2
56  slt     $15, $6, $7
            ...
72  lw      $4, 50($7)
```

Organização do MIPS: pipeline

- Conflito de controle

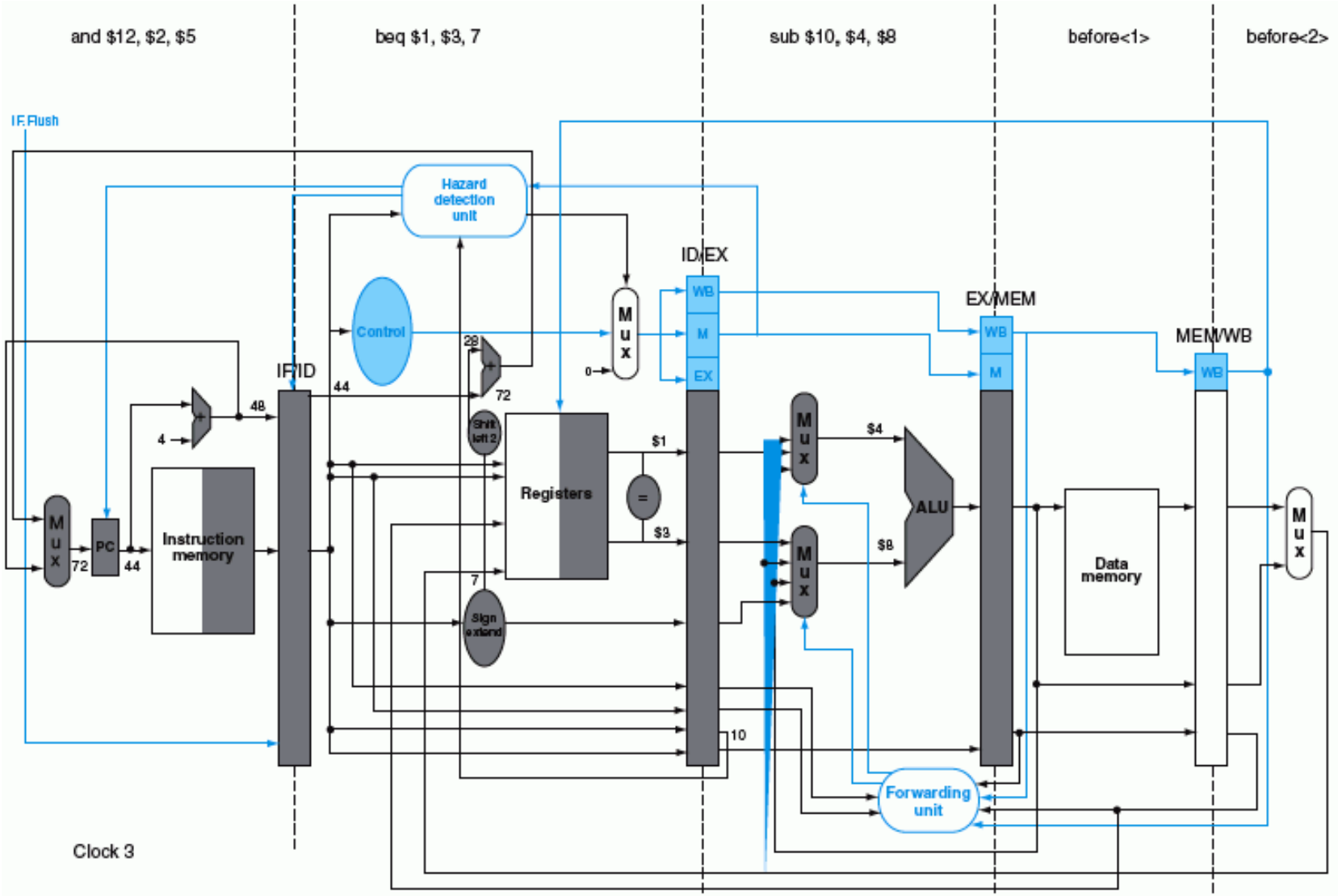
  Considerando o bloco operativo pipeline visto



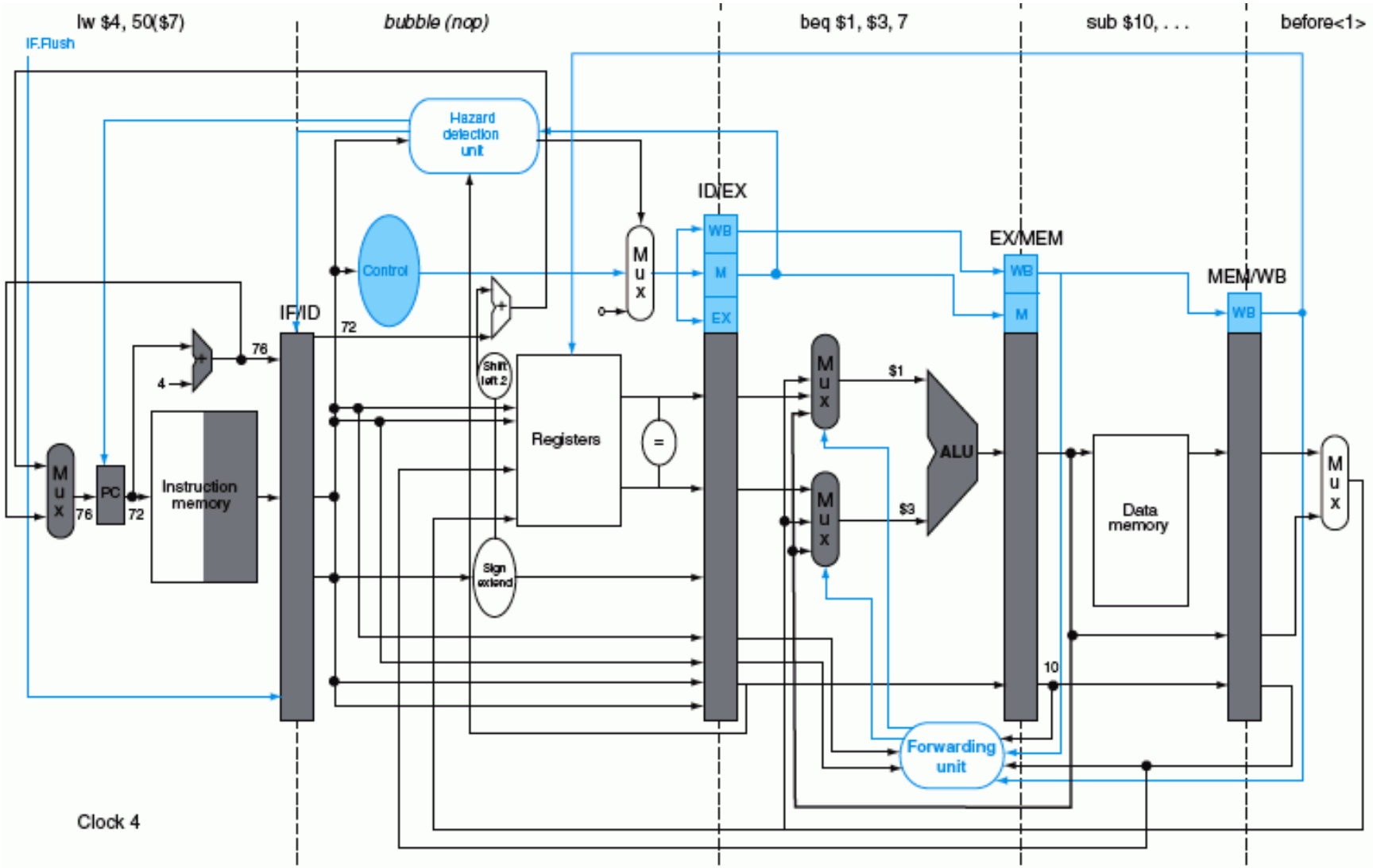
Consideração estática

- Antecipação do desvio
  - Apesar das dificuldades reduz a penalidade
    - Apenas uma instrução é eliminada se desvio for tomado
    - Instrução presente no estágio BI é descartado
  - Descarte da instrução exige a inclusão de um único sinal de controle
    - BI.*flush*
    - Transforma a instrução em BI em NOP

Antecipação do desvio (BEQ em DI)



Antecipação do desvio (BEQ em EX)



Previsão dinâmica de desvios

- Previsão estática é adequada para arquiteturas que exploram pouca profundidade em pipelines
  - O erro na previsão implica em custo relativamente baixo quando comparado a arquiteturas de pipeline mais profundo

- Previsão dinâmica de desvios leva em consideração decisões passadas
  - Associa um dado endereço a instrução de desvio
  - Caso um desvio tenha ocorrido previamente o novo endereço de PC é o mesmo assumido anteriormente, e não PC+4

Previsão dinâmica de desvios

- Proposta de implementação
  - Buffer de previsão de desvios (tabela de histórico de desvios)
    - Implementação feita em um bit
      - Último salto ocorreu ou não
    - Organização da memória de histórico
      - Indexada pelos parte menos significativa do endereço do PC da instrução de desvio
      - Contém um bit referenciando se um desvio foi realizado recentemente ou não
    - Proposta simples que não garante a corretude da previsão
      - Bit de salto pode ter sido assinalida por outra instrução
    - Previsão incorreta reflete em tratamento
      - Estratégia similar de descarte tem de ser implementado como visto antes também

Previsão dinâmica de desvios

- Outra proposta de implementação
  - Implementação feita em dois bit
    - Explora frequência de desvios regulares
    - Esquema mais empregado quando comparado com a implementação de 1 bit
    - Duas decisões errôneas têm de ser assinalada para efetivar a atualização da decisão correta

Mecanismo de previsão dinâmica

- Estados de um esquema de previsão de 2 bits