Pipeline

Luciano L. Caimi

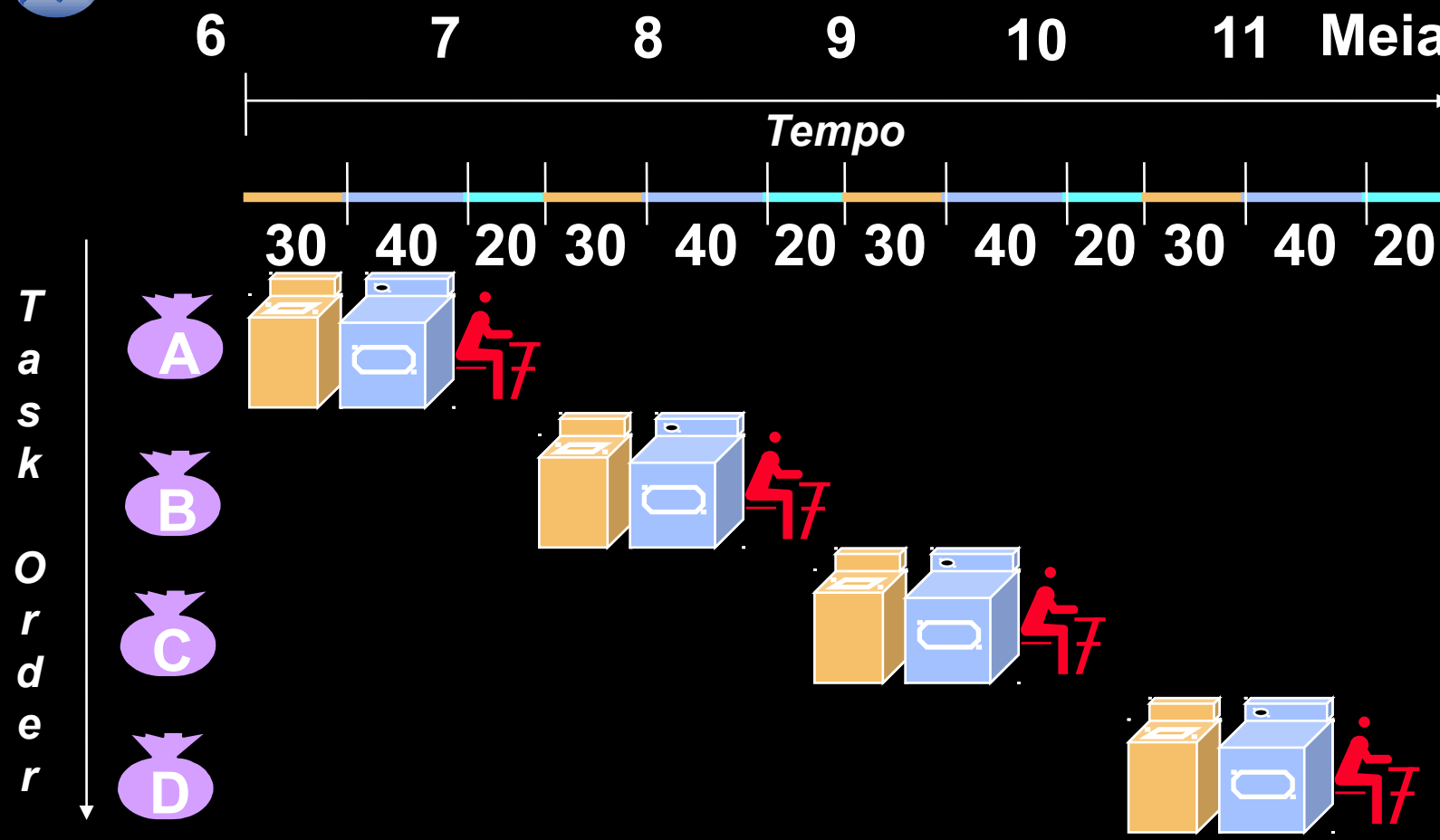
lcaimi@uffs.edu.br

Pipelines

Sumário

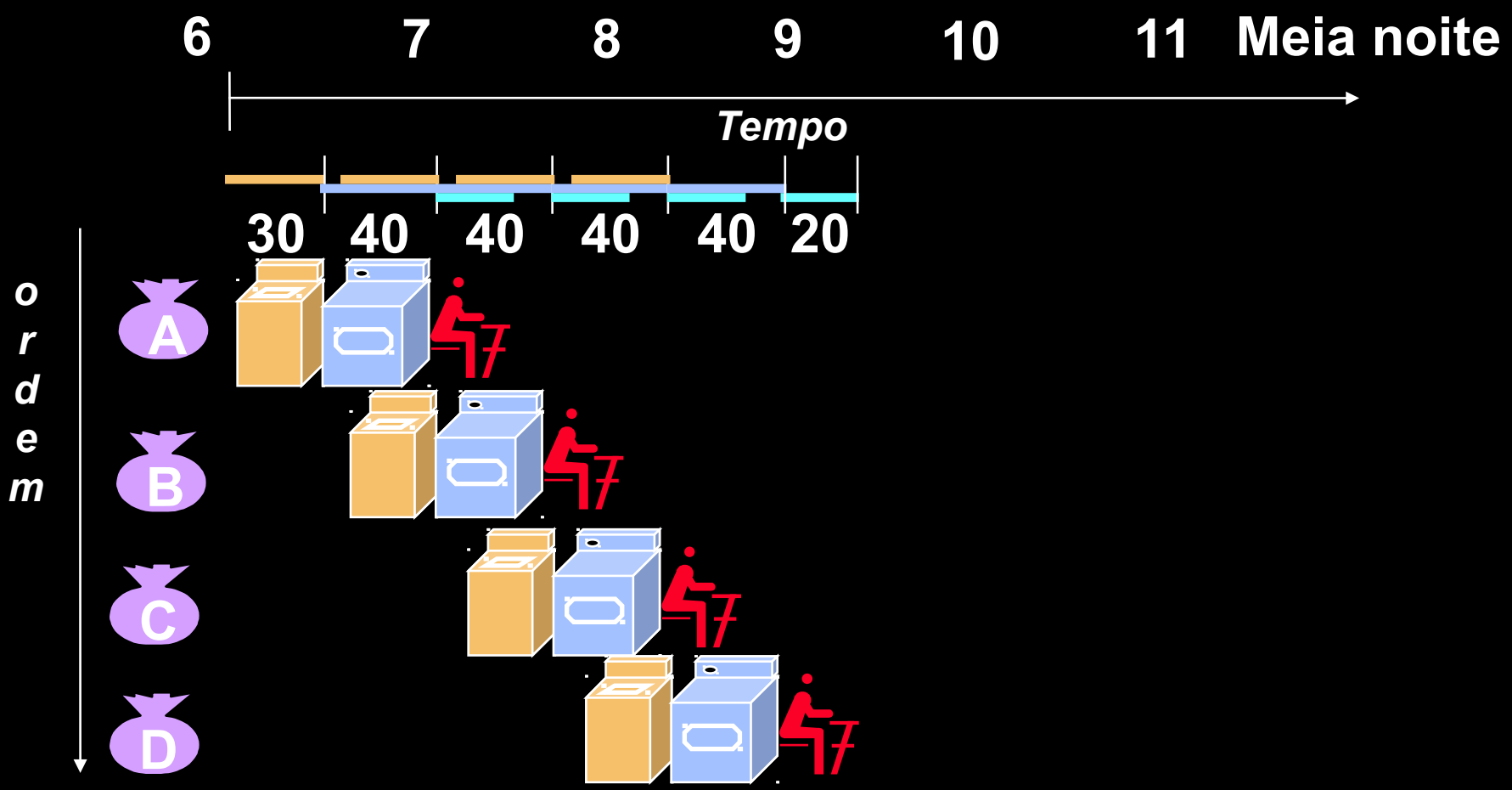
1. Introdução
2. Pipelines de instruções
3. Medidas
4. Conflitos de memória
5. Dependências em desvios
6. Dependências de dados

Introdução



- A lavanderia sequencial leva 6 horas para 4 volumes
- Se usarem o "pipeline", quanto tempo

Introdução

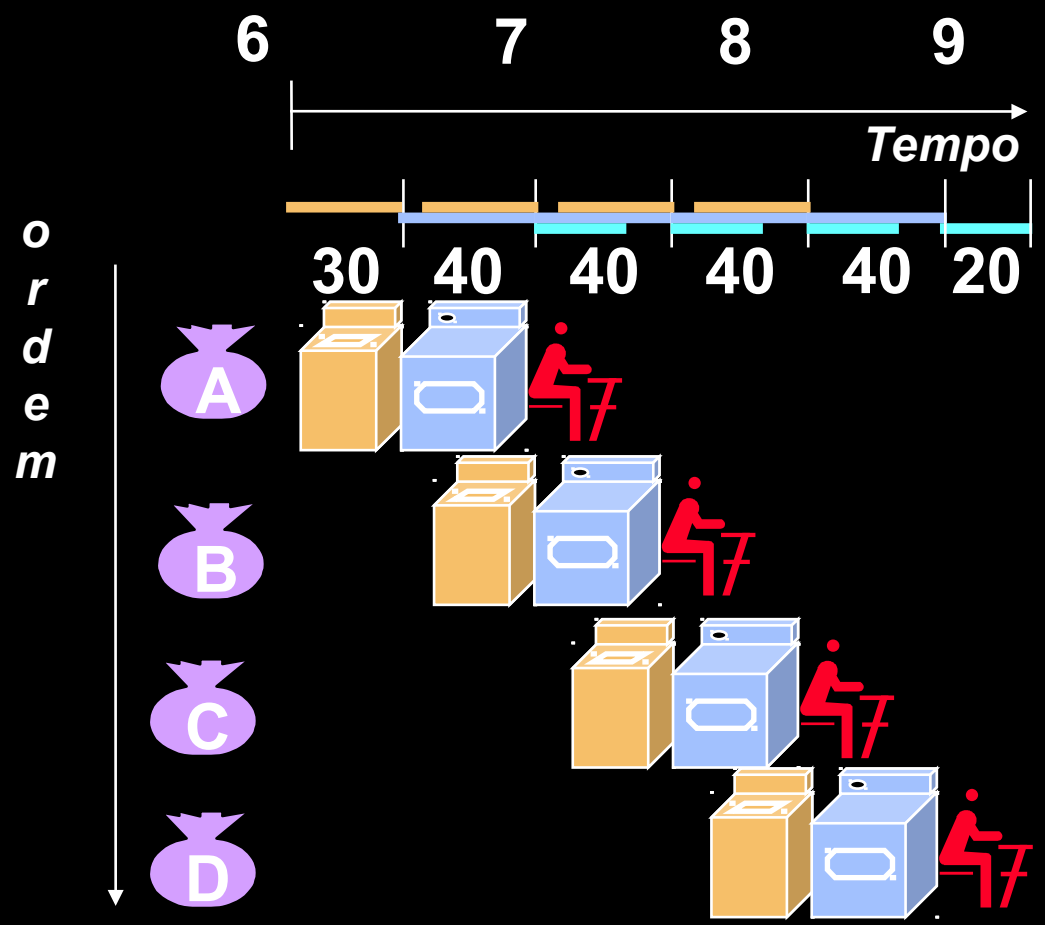


- **Lavanderia em Pipeline leva 3.5 horas**

Introdução

- **Objetivo: aumento de desempenho**
  - **divisão de uma tarefa em N estágios**
  - **N tarefas executadas em paralelo, uma em cada estágio**

Introdução



- O Pipeline não ajuda melhorar a **latência** de uma única tarefa, ele ajuda no **throughput** de um trabalho por completo
- A taxa do Pipeline é limi-tada pelo estágio **mais lento**
- **Multiplas** tarefas operam simultaneamente
- Speedup ideal = **Número de estágios**

Introdução

- O tempo para “preencher” o pipeline e o tempo para “limpar” o pipeline reduzem o speedup
- Executa bilhões de instruções, tal que o importante seja o throughout
- Fatores desejados:

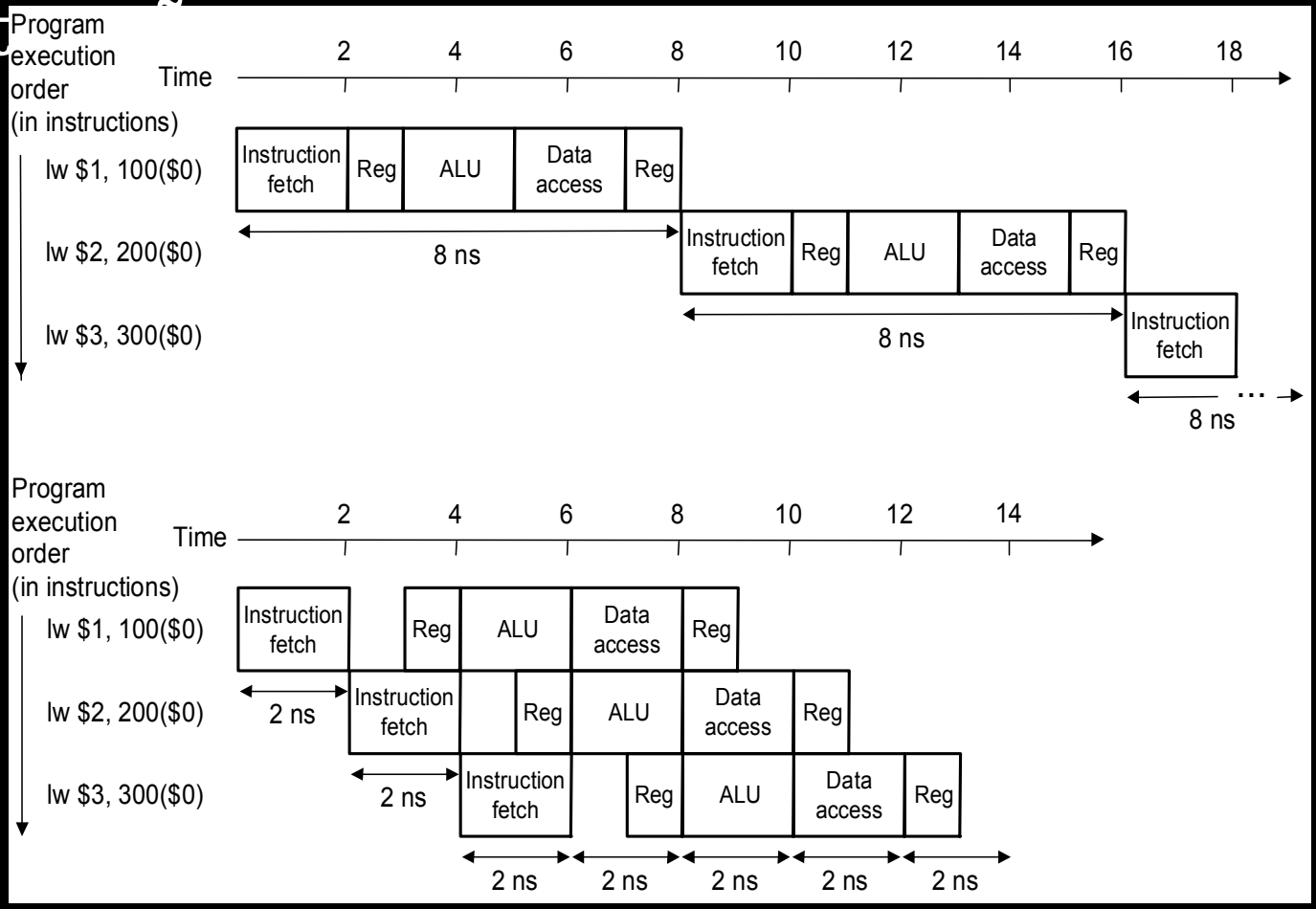
  - todas as instruções tenham o mesmo comprimento,

  - os campos de registradores sejam localizados numa mesma posição no formato de instrução,

  - referências à memória somente através de instruções loads ou stores

Introdução

- O pipeline melhora o desempenho aumentando o throughput de inst

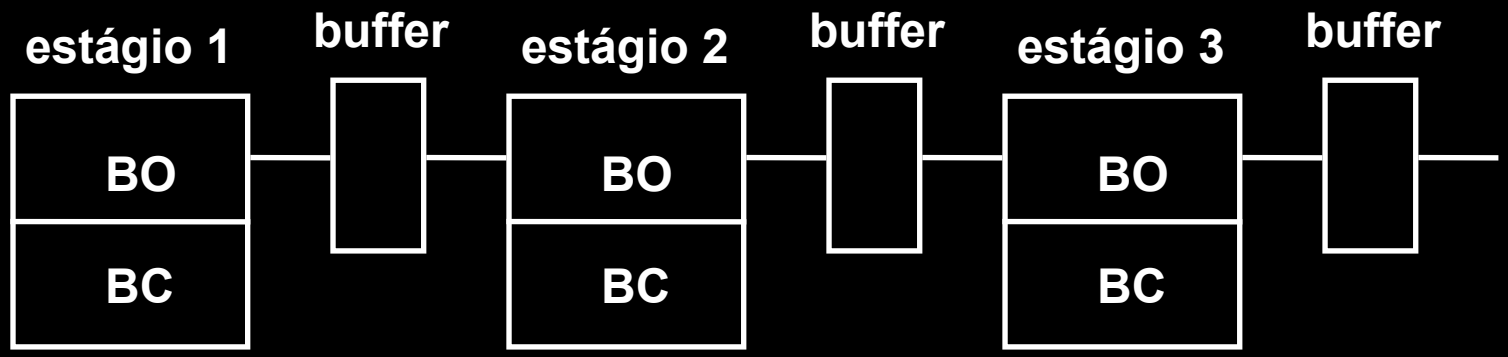


Introdução

- O que o torna fácil
  - Todas as instruções com mesmo comprimento
  - Somente poucos formatos de instruções
  - Os operandos de memória aparecem somente em loads e stores
- O que o torna difícil?
  - conflitos estruturais: supor que temos somente uma memória
  - conflitos de controle: preocupar com instruções de branch
  - conflitos de dados: uma instrução depende de uma instrução prévia
- Veremos os modernos processadores e verificar o que realmente os torna difíceis de serem implementados:

Introdução

- **bloco operacional e bloco de controle independentes para cada estágio**
- **necessidade de buffers entre os estágios**



Classificações

- **Baseada nos níveis de processamento**
  - **Pipeline de instruções: mais comum dos tipos de pipeline, baseada na divisão dos estágios do ciclo de instrução;**
  - **Pipeline Aritmético: a ULA é segmentada em vários blocos para execução das operações;**
- **Baseada na configuração e estratégia de controle**
  - **Unifunção x Multifunção: pipeline dedicados ou pipelines reconfiguráveis;**
  - **Estático x Dinâmico: decorrente da classificação anterior;**
  - **Escalar x Vetorial: quando projetado para instruções escalares ou instruções vetoriais**

Pipelines de instruções

- pipeline com 2 estágios: fetch + execução
- supondo instrução R1 = R1 + R2
- fetch

  REM = PC

  ler memória: RDM = MEM [REM]

  PC = PC + 1

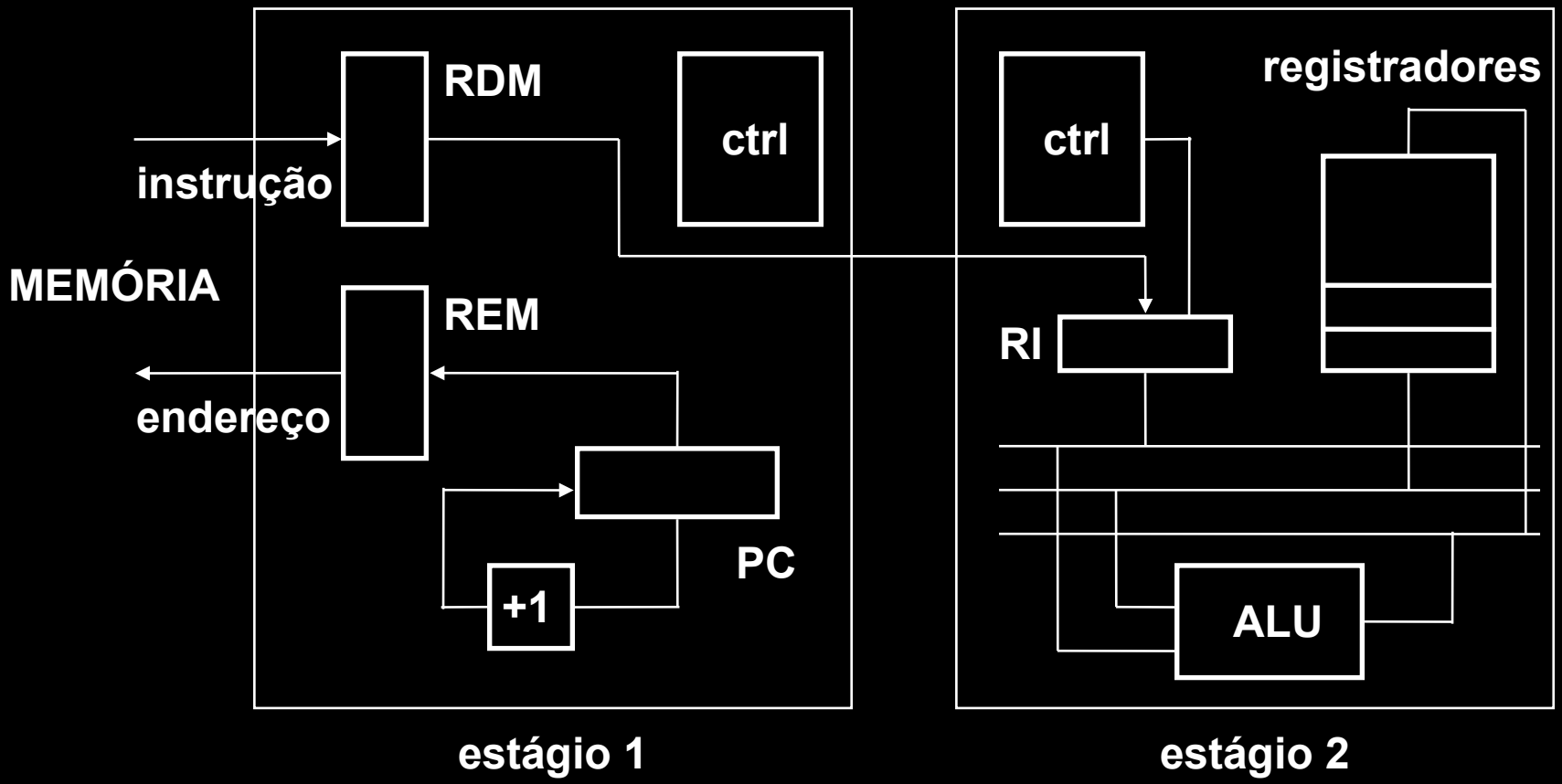
  RI = RDM
- execução

  a = R1

  b = R2

  R1 = a + b

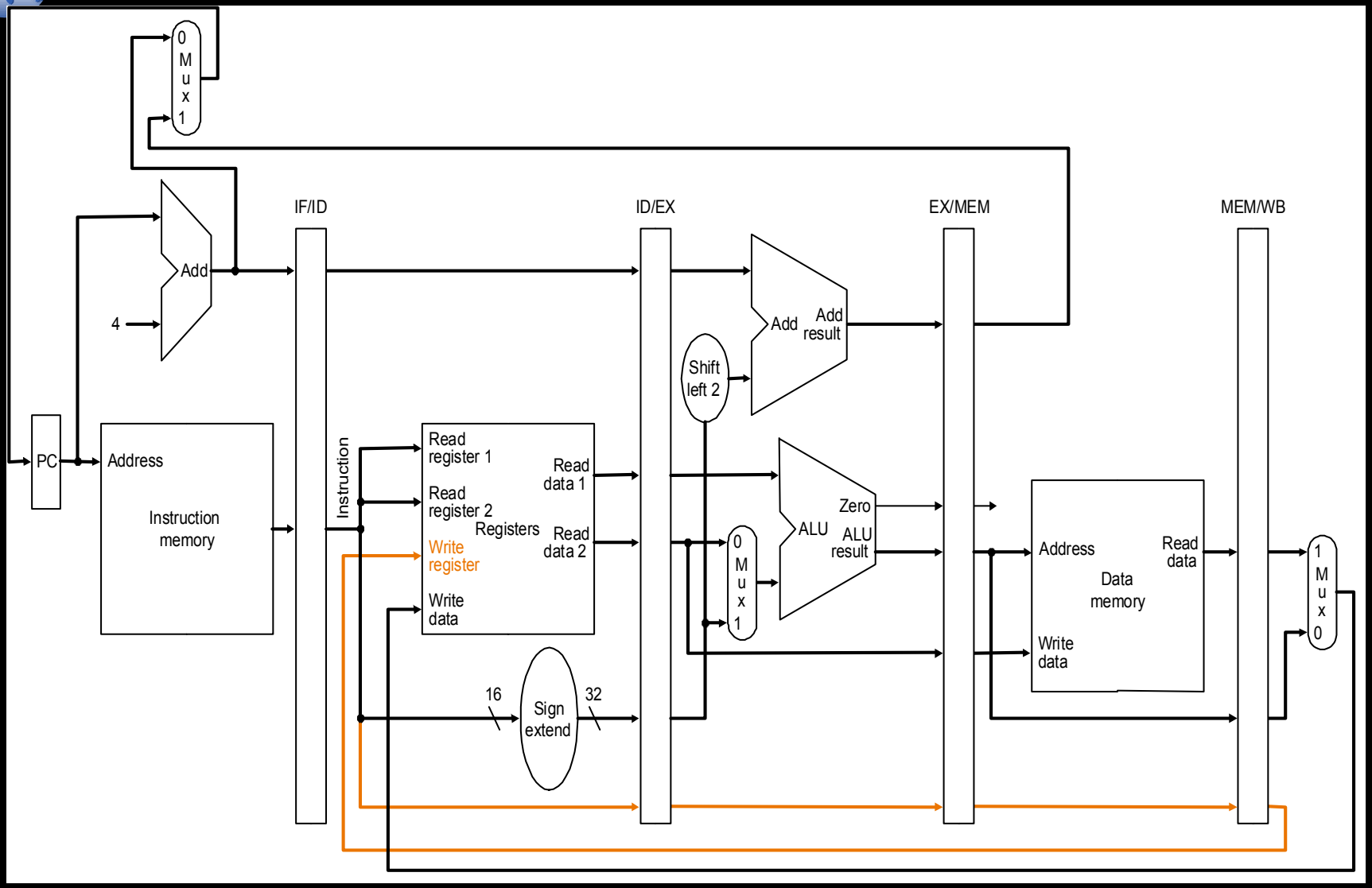
Pipeline com 2 estágios



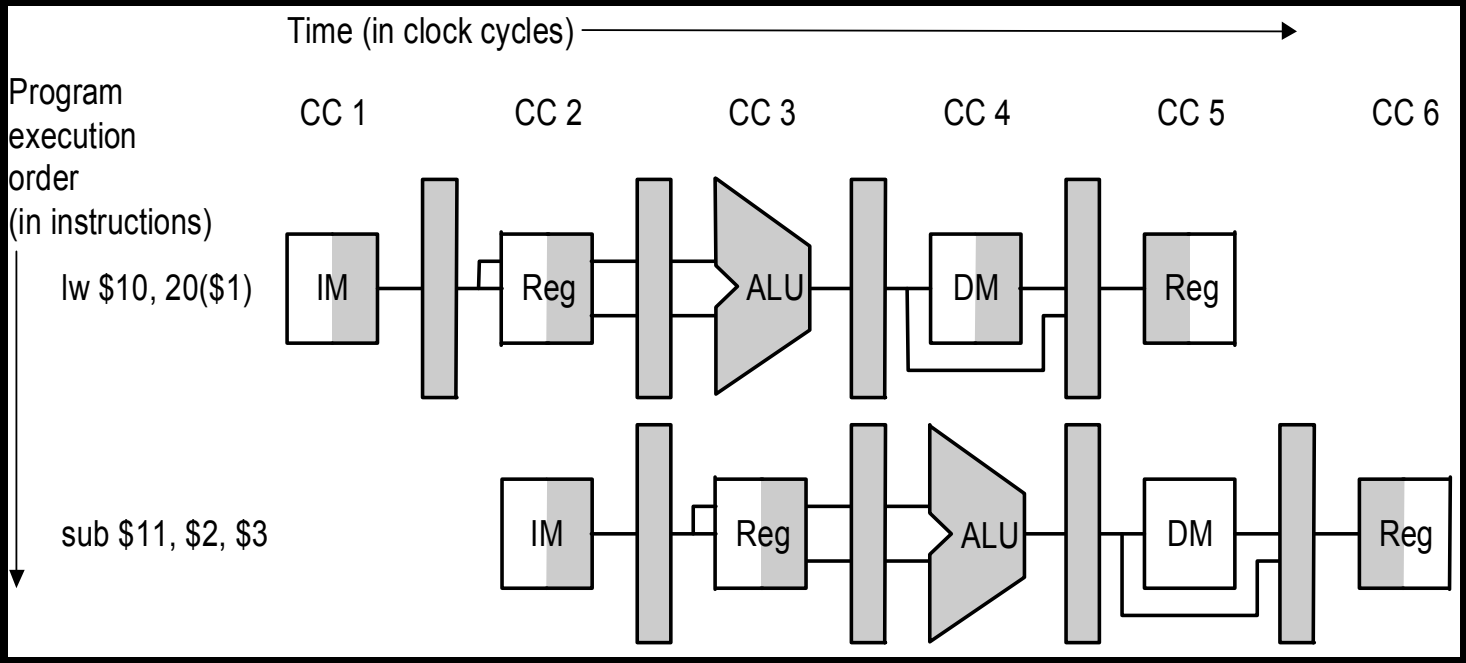
Pipelines com mais estágios

- 3 estágios
  - fetch, decodificação / busca de operandos, execução
- 4 estágios
  - fetch, decodificação / busca de operandos, execução, store
- 5 estágios
  - fetch, decodificação / cálculo de endereço de operandos, busca de operandos, execução, store
- 6 estágios
  - fetch, decodificação, cálculo de endereço de operandos, busca de operandos, execução, store
  - estágio só para decodificação é bom em processadores CISC

Pipelines de instruções

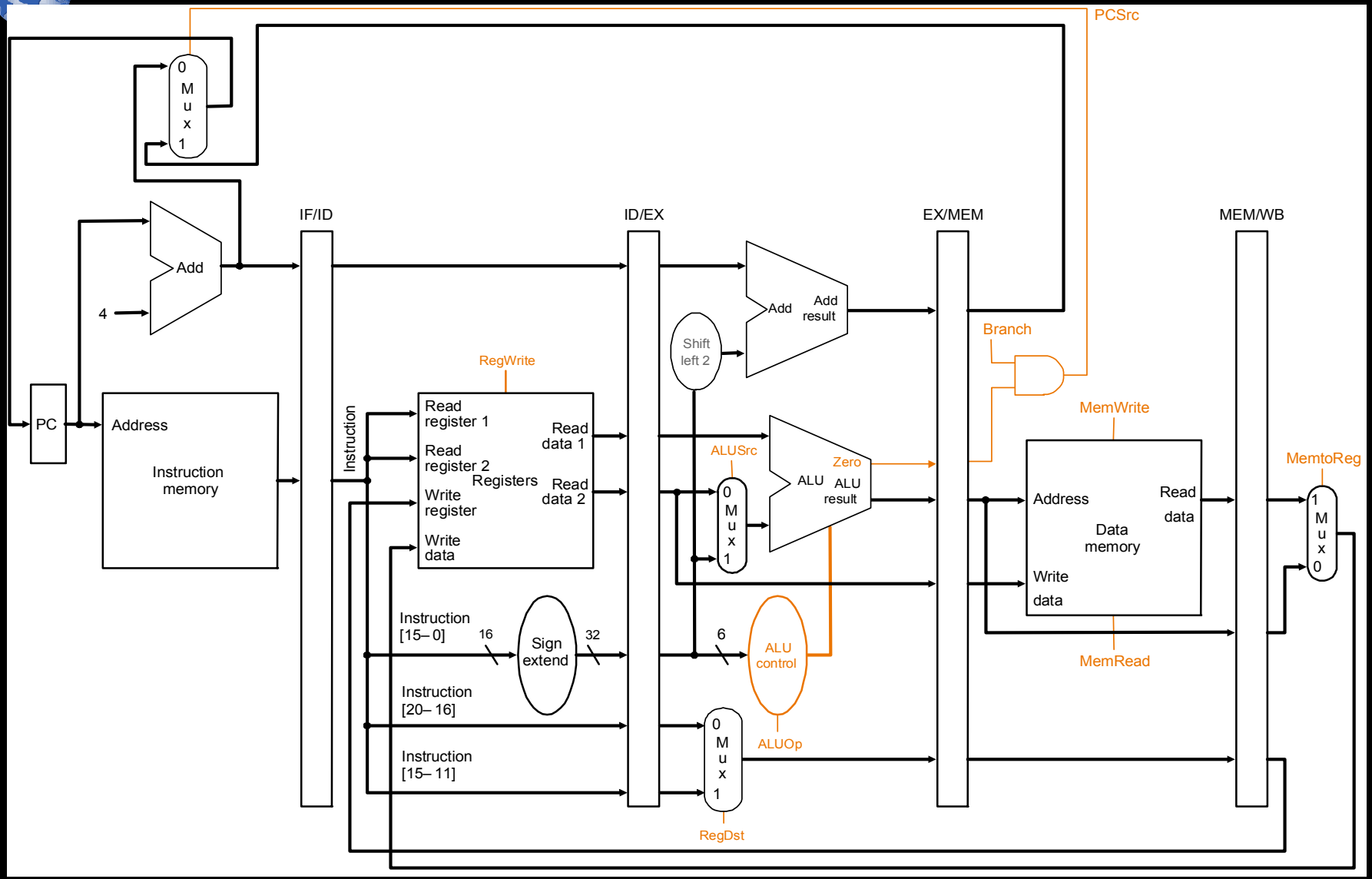


Pipelines representados graficamente



- **Pode nos ajudar a responder questões como:**
  - **Quantos ciclos leva para executar este código?**
  - **O que a ALU estaria fazendo durante o ciclo 4?**
  - **Usar esta representação para nos auxiliar na compreensão do fluxo de dados**

Controle do Pipeline



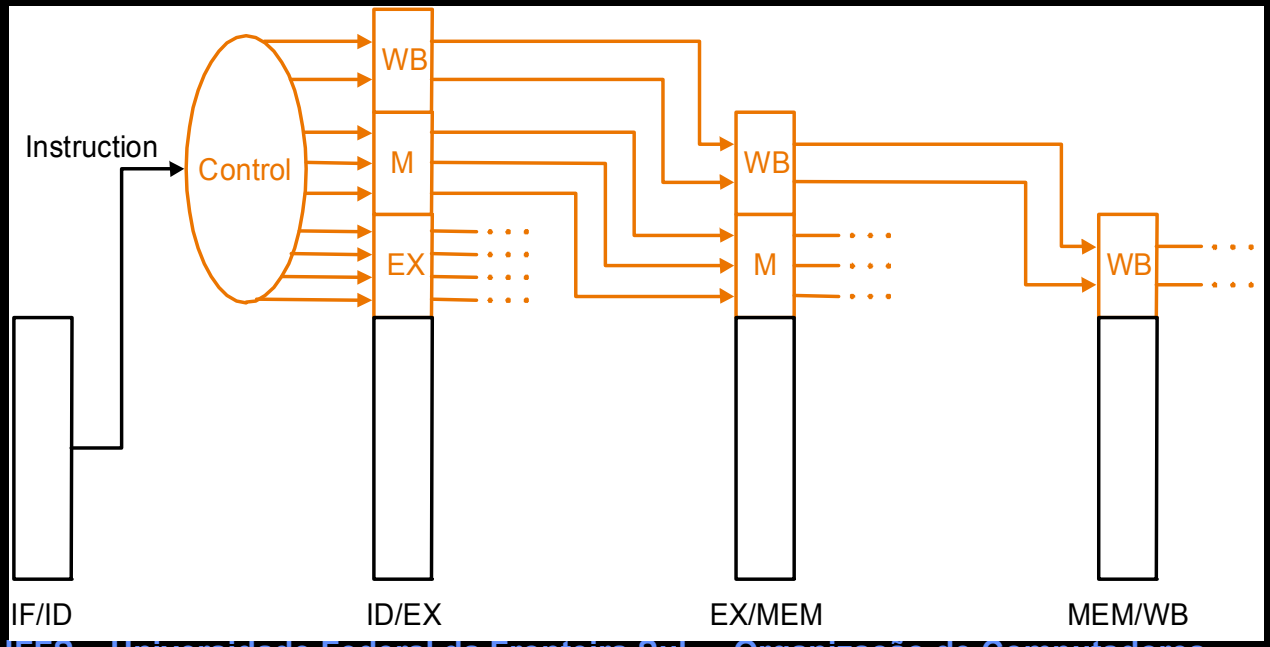
Controle do Pipeline

- **Temos 5 estágios. O que necessita ser controlado em cada estágio?**
  - **Busca de Instrução e Incremento do PC**
  - **Decodificação da Instrução / Busca de Registradores**
  - **Execução**
  - **Estágio de Memória**
  - **Write Back**
- **Como seria controlada uma fábrica de automóveis?**
  - **Uma central de controle dizendo o que cada um deve fazer?**
  - **Usaríamos uma máquina de estado finito?**

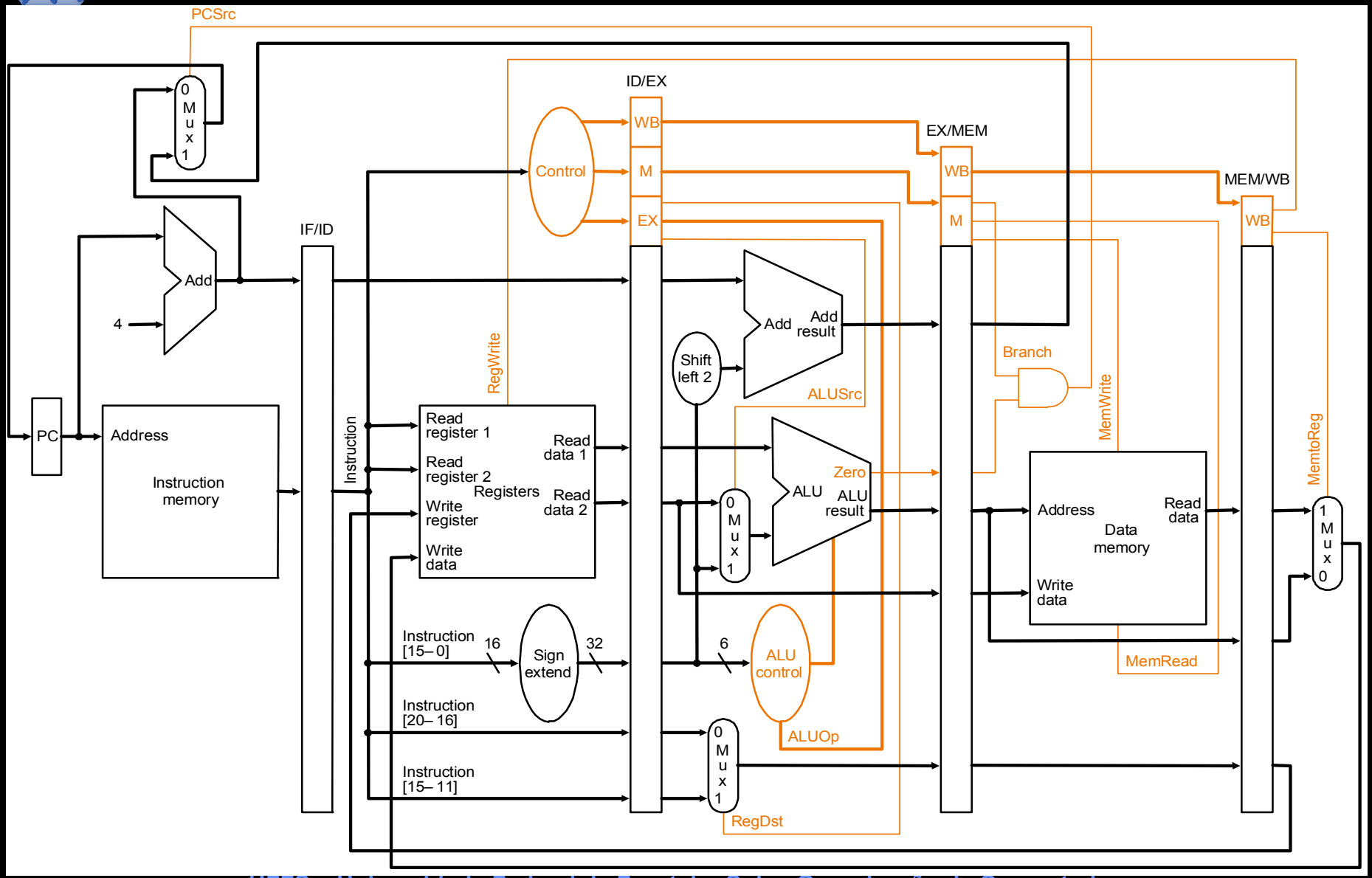
Controle do Pipeline

Passar os sinais de controle como os dad

Instruction	Execution/Address Calculation stage control lines				Memory access stage control lines			Write-back stage control lines	
	Reg Dst	ALU Op1	ALU Op0	ALU Src	Branch	Mem Read	Mem Write	Reg write	Mem to Reg



Fluxo de dados com Controle



Medidas

- existe um tempo inicial até que o pipeline “encha”
- cada tarefa leva o mesmo tempo, com ou sem pipeline
- média de tempo por tarefa é no entanto dividida por K, sendo k o número de estágios.

Medidas

O tempo de ciclo τ de um *pipeline* de instrução é o tempo requerido para avançar um conjunto de instruções um estágio. O tempo de ciclo pode ser determinado da seguinte maneira:

onde:

τ_m = atraso máximo de estágio

k = número de estágios do *pipeline* de instrução

d = tempo necessário para propagar sinais e dados de um estágio para o próximo

Em geral, d é equivalente ao pulso de um relógio e $\tau_m >> d$.

Medidas

Suponha que sejam processadas *n* instruções, sem que ocorra desvio. O tempo total de execução é dado por:

O *speedup* para a execução com o *pipeline* de instruções em relação à execução sem o uso do *pipeline* é:

Em função do número de instruções executadas sem desvio, o fator de aceleração é igual a *k* quando $n \rightarrow \infty$.

Medidas

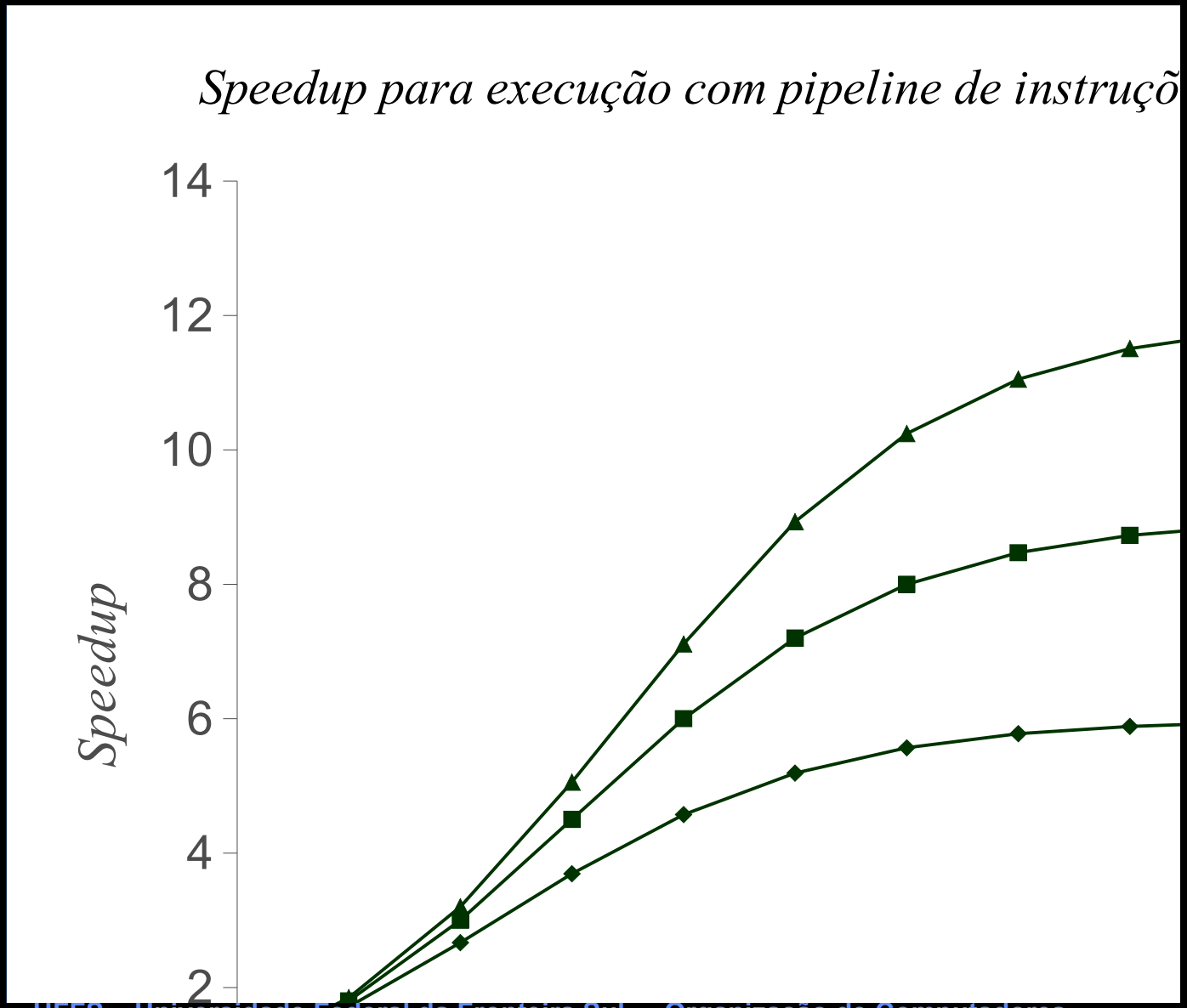
Em função do número de estágios, o fator de aceleração se aproxima do número de instruções que podem ser introduzidas no *pipeline* sem desvio.

Quanto maior o número de estágios do *pipeline*, maior o *speedup*. No entanto, o ganho diminui devido:

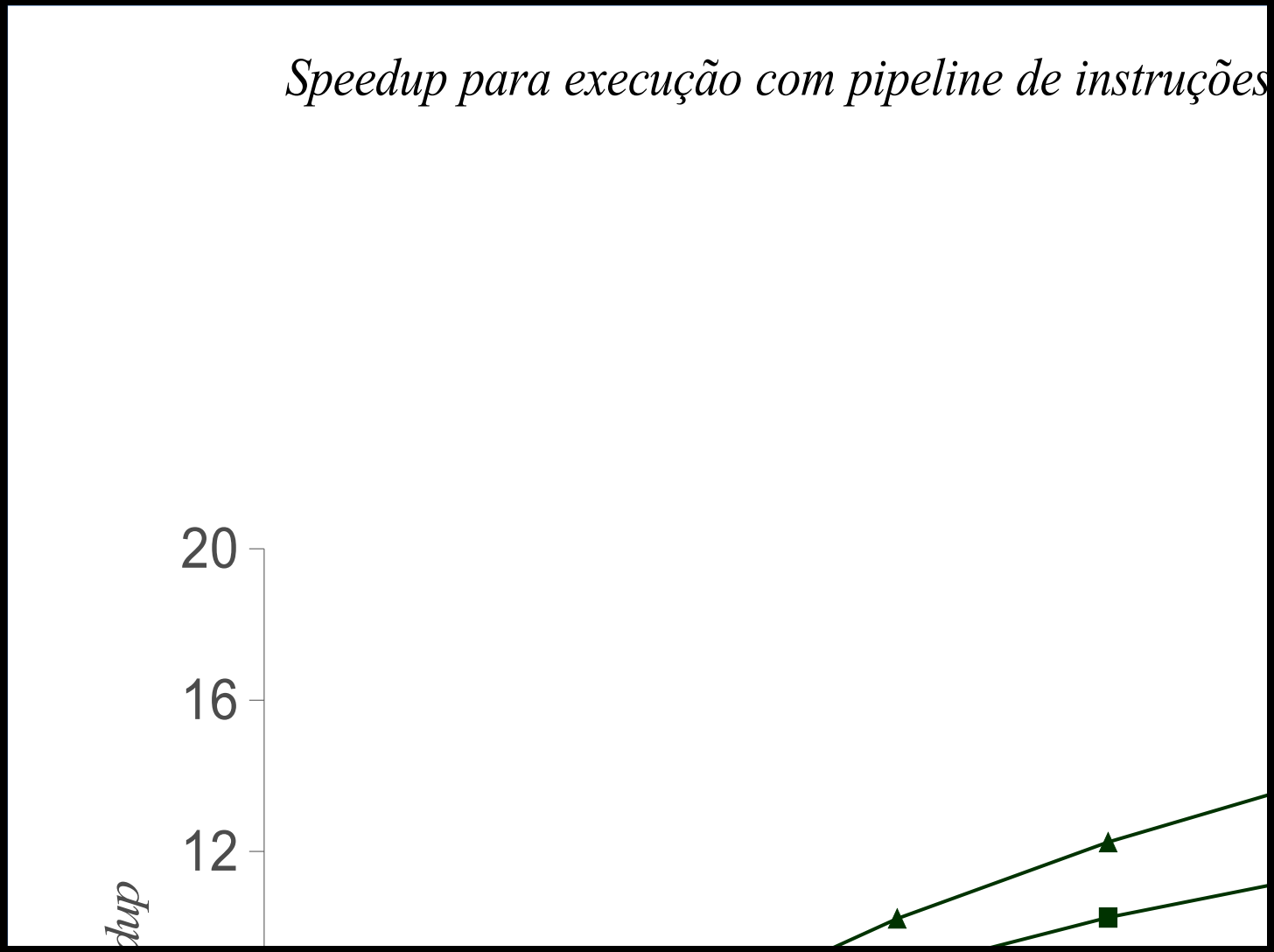
- ao aumento no custo da implementação;
- aos atrasos entre estágios;
- aos atrasos no processo de esvaziamento do *pipeline* quando ocorre instrução de desvio.

Um número de estágios entre 6 e 9 parece ser mais adequado.

Medidas


Speedup para execução com pipeline de instruçõ
Speedup
14
12
10
8
6
4
2

Medidas

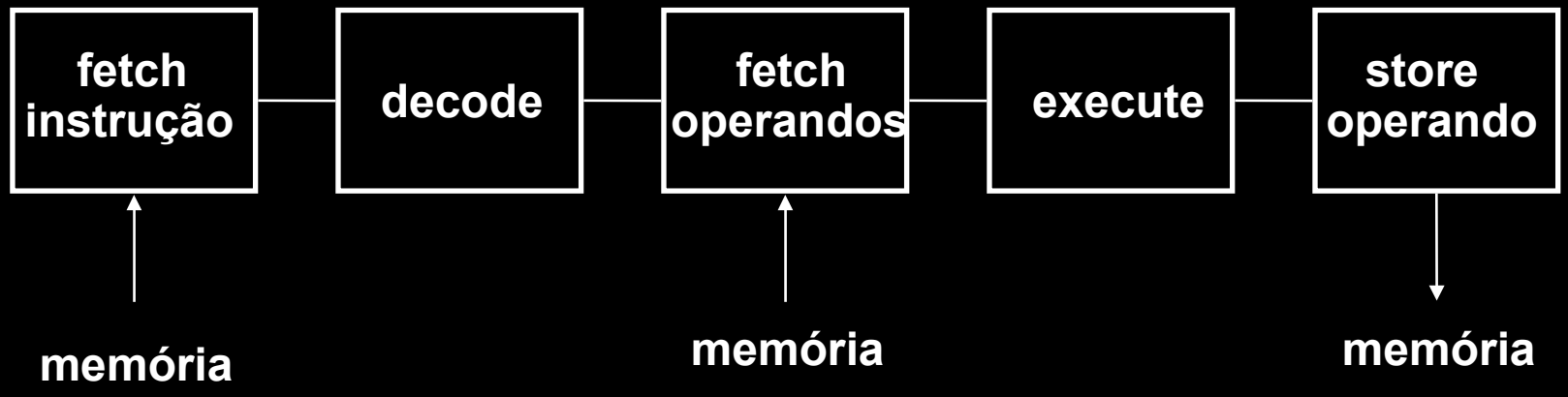


Problemas no desempenho

- como dividir todas as instruções num mesmo conjunto de estágios?
- como obter estágios com tempos de execução similares?
- conflitos de memória
  - acessos simultâneos à memória por 2 ou mais estágios
- dependências de dados
  - instruções dependem de resultados de instruções anteriores, ainda não completadas
- instruções de desvio
  - instrução seguinte não está no endereço seguinte à instrução anterior

4. Conflitos de memória

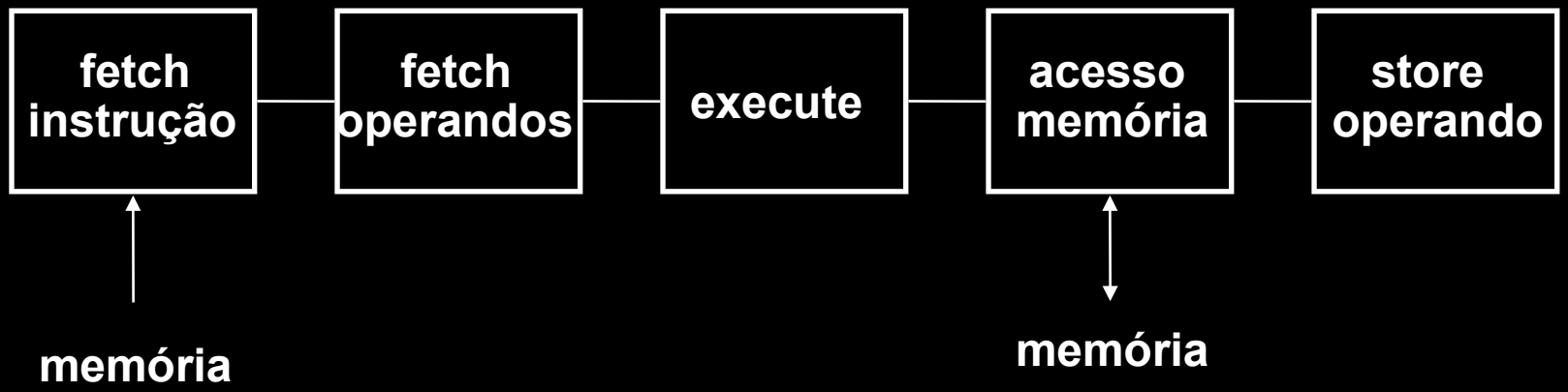
- problema: acessos simultâneos à memória por 2 ou mais estágios



- soluções
  - caches separadas de dados e instruções
  - memória entrelaçada

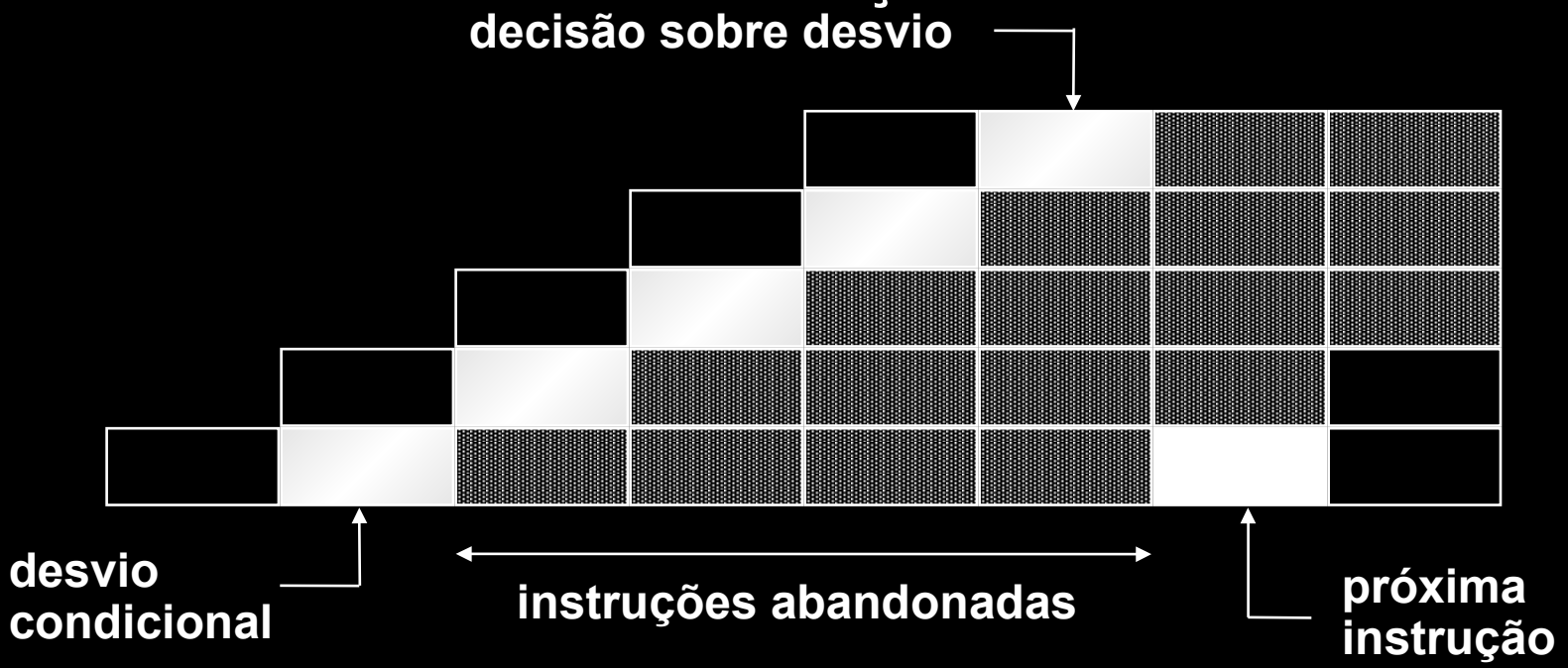
Processadores RISC

- memória é acessada apenas por instruções LOAD e STORE
- apenas um estágio do pipeline faz acesso a operandos de memória
- apenas 1 instrução pode estar executando acesso a dados a cada instante



5. Dependências em desvios

- efeito de desvios condicionais
  - se o desvio ocorre, pipeline precisa ser esvaziado
  - não se sabe se desvio ocorrerá ou não até o momento de sua execução



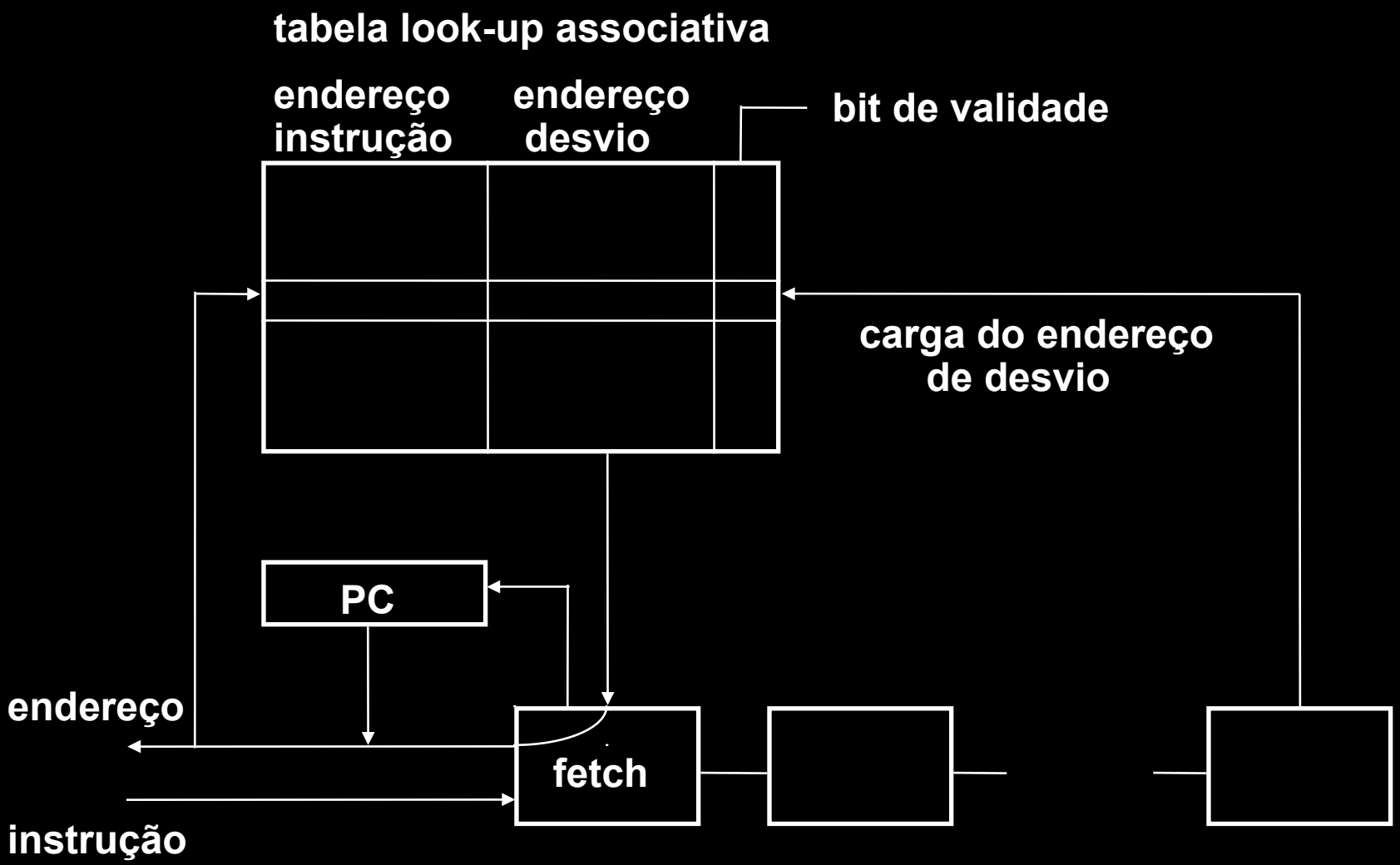
Dependências em desvios

- instruções abandonadas não podem ter afetado conteúdo de registradores e memórias
- desvios incondicionais
  - sabe-se que é um desvio desde a decodificação da instrução (segundo estágio do pipeline)
  - é possível evitar abandono de número maior de instruções
  - problema: em que estágio é feito o cálculo do endereço efetivo do desvio?

Predição estática

- supor sempre mesma direção para o desvio
  - desvio sempre ocorre
  - desvio nunca ocorre
- compilador define direção mais provável
  - instrução de desvio contém bit de predição, ligado / desligado pelo compilador
  - início de laço (ou desvio para frente): desvio improvável
  - final de laço (ou desvio para trás): desvio provável
- até 85 % de acerto é possível

Predição dinâmica



Predição dinâmica

- tabela look-up associativa armazena pares
  - endereços das instruções de desvio condicional mais recentemente executadas
  - endereços de destino destes desvios
- quando instrução de desvio condicional é executada
  - é feita comparação associativa na tabela, à procura do endereço desta instrução
  - se endereço é encontrado e bit de validade está ligado, o endereço de desvio armazenado na tabela é usado
  - ao final da execução da instrução, endereço efetivo de destino do desvio é armazenado na tabela
- tabela pode utilizar diversos mapeamentos e algoritmos de substituição

Predição dinâmica

- variação: *branch history table*
  - contador associado a cada posição da tabela
  - a cada vez que uma instrução de desvio contida na tabela é executada ...
    - contador é incrementado se desvio ocorre
    - contador é decrementado se desvio não ocorre
  - valor do contador é utilizado para a predição

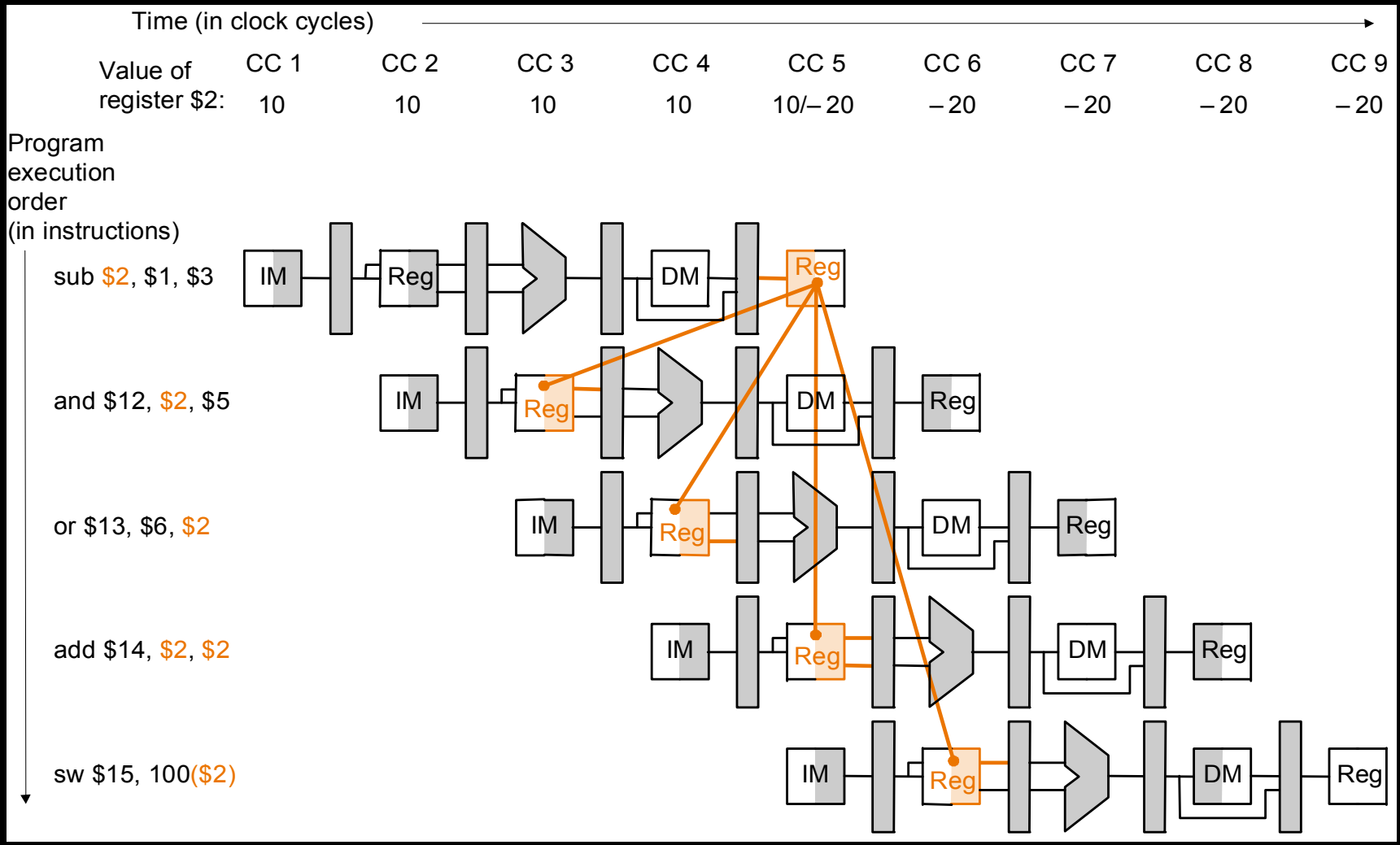
6. Dependências de dados

- problema: instruções consecutivas podem fazer acesso aos mesmos operandos
- caso particular: instrução precisa de resultado anterior (p.ex. registrador) para cálculo de endereço efetivo de operando
- tipos de dependências de dados
  - dependência verdadeira (read-after-write)
  - Antidependência (write-after-read)
  - dependência de saída (write-after-write)

Dependência verdadeira

- **exemplo**
  1. **ADD R3, R2, R1 ; R3 = R2 + R1**
  2. **SUB R4, R3, 1 ; R4 = R3 – 1**
- **instrução 2 depende de valor calculado pela instrução 1**
- **instrução 1 precisa ser completada antes que instrução 2 busque os operandos**
- ***read-after-write hazard***
- **pipeline precisa ser parado durante certo número de ciclos ou um forwarding deve acontecer**

Dependências



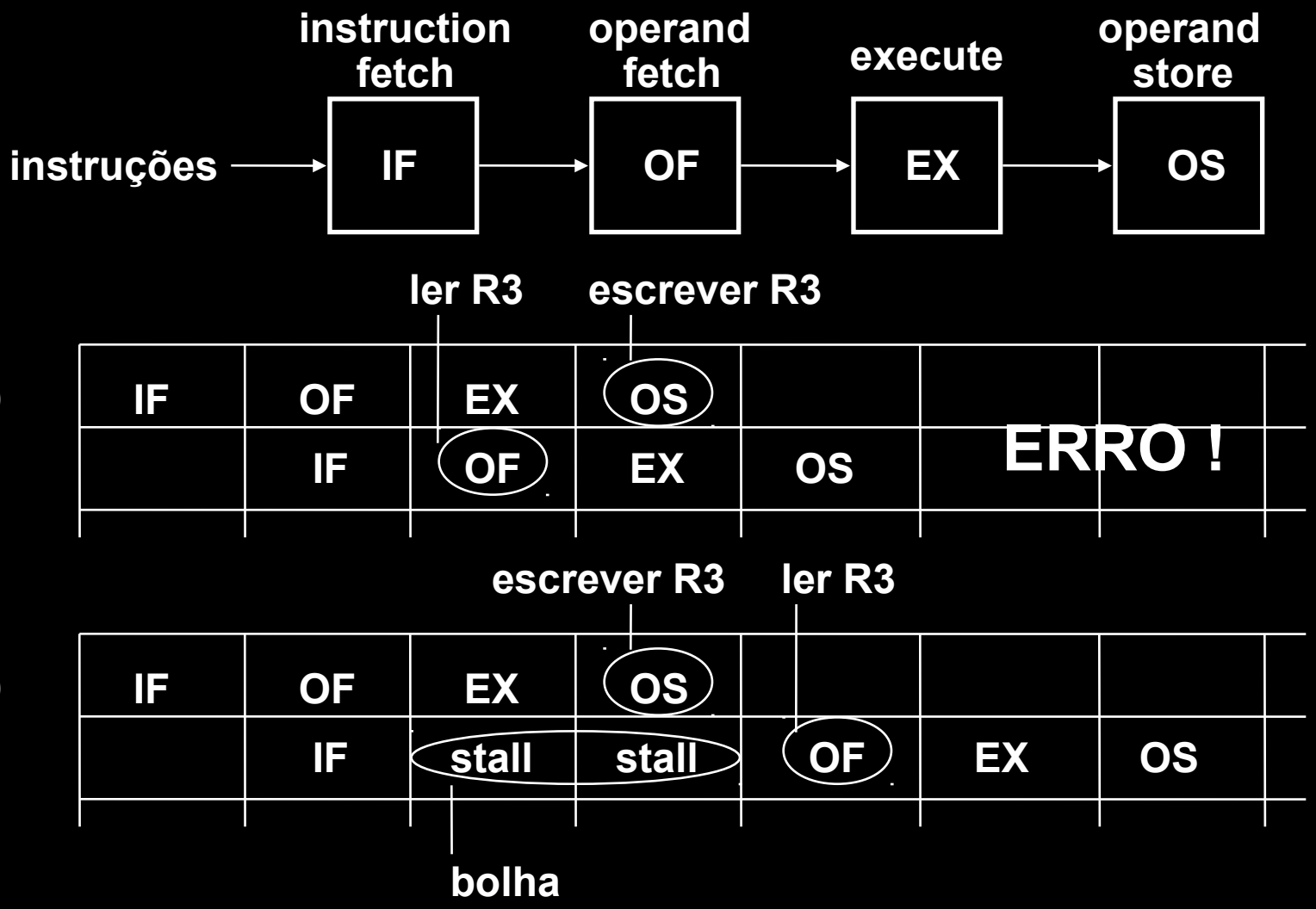
Solução de Software

- O compilador deve garantir que não haja conflitos
- Onde inserir os “nops” ?

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

Problema: isso torna o computador lento!

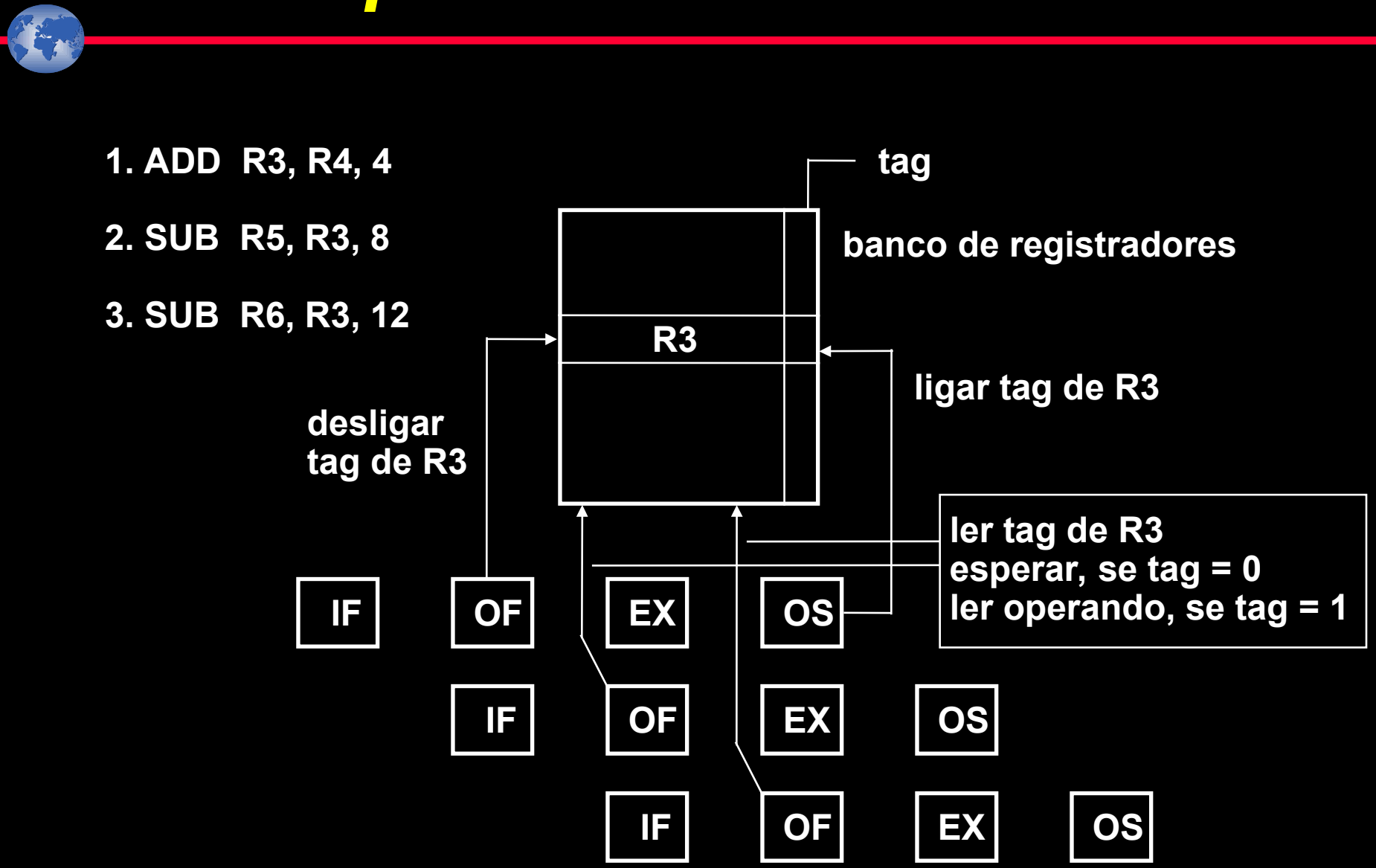
Dependência verdadeira



Pipeline interlock

- método de manter seqüência correta de leituras e escritas em registradores
- tag de 1 bit é associado a cada registrador
  - tag = 0 indica valor não válido, = 1 indica valor válido
- se instrução que é buscada e decodificada escreve num registrador, tag do mesmo é zerado
- tag é ligado quando instrução escreve o valor no registrador
- outras instruções que sejam executadas enquanto tag está zerado devem esperar tag = 1 para ler valor deste registrador

Pipeline interlock



Antidependência

- **exemplo**
  1. **ADD R3, R2, R1 ; R3 = R2 + R1**
  2. **SUB R2, R4, 1 ; R2 = R4 – 1**
- **instrução 1 utiliza operando que é escrito pela instrução 2**
- **instrução 2 não pode salvar resultado antes que instrução 1 tenha lido seus operandos**
- ***write-after-read hazard***
- **não é um problema em pipelines onde a ordem de execução das instruções é mantida**
  - **escrita do resultado é sempre o último**

Dependência de saída

- **exemplo**
  1. **ADD R3, R2, R1 ; R3 = R2 + R1**
  2. **SUB R2, R3, 1 ; R2 = R3 – 1**
  3. **ADD R3, R2, R5 ; R3 = R2 + R5**
- **instruções 1 e 3 escrevem no mesmo operando**
- **instrução 1 tem que escrever seu resultado em R3 antes que a instrução 3, senão instrução 2 usará valor errado de R3**
- ***write-after-write hazard***
- **também só é problema em processadores super-escalares**

Forwarding (antecipação)

- exemplo

  ADD R3, R2, R0

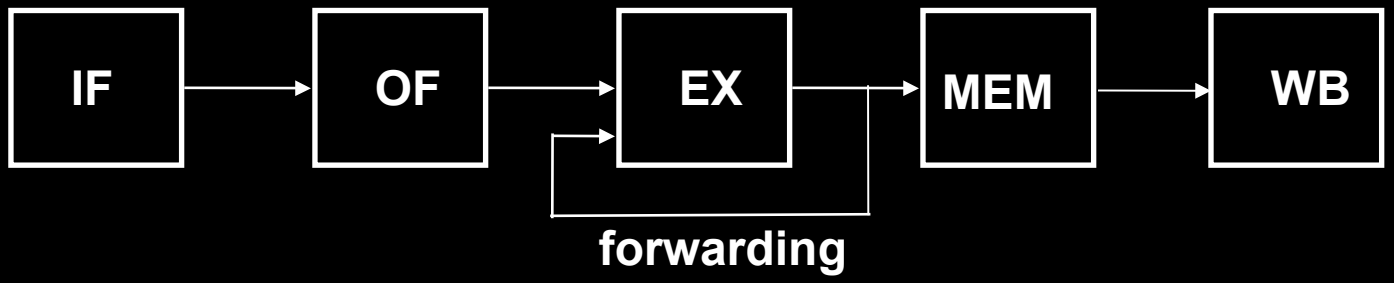
  SUB R4, R3, 8

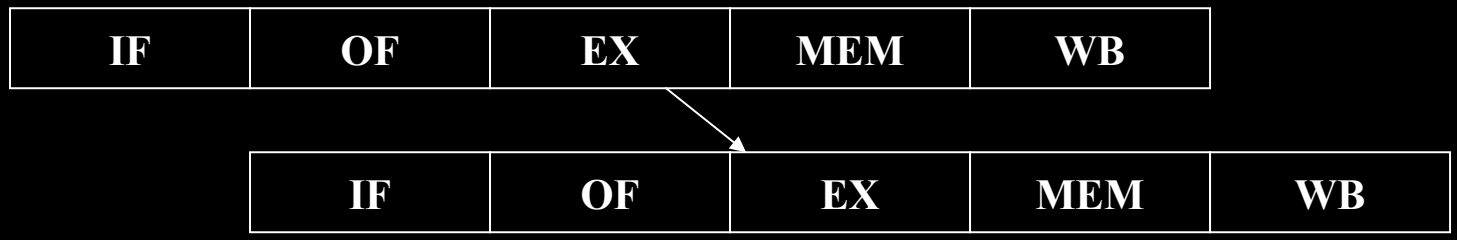
- SUB precisa do valor de R3, calculado pelo ADD
  - valor é escrito em R3 por ADD no último estágio WB (write-back)
  - valor é necessário em SUB no terceiro estágio
  - instrução SUB ficará presa por 2 ciclos no 2º estágio do pipeline

supõe-se escrita no banco de registradores na primeira metade do ciclo e leitura na segunda metade

Forwarding

- **caminho interno dentro do pipeline entre a saída da ALU e a entrada da ALU**
  - **evita *stall* do pipeline**





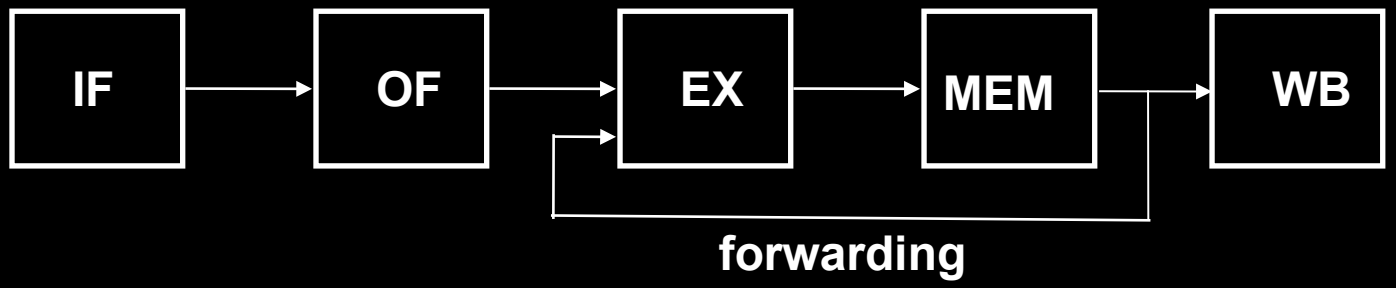
Forwarding

- exemplo 2

  LOAD R3, 100 (R0)

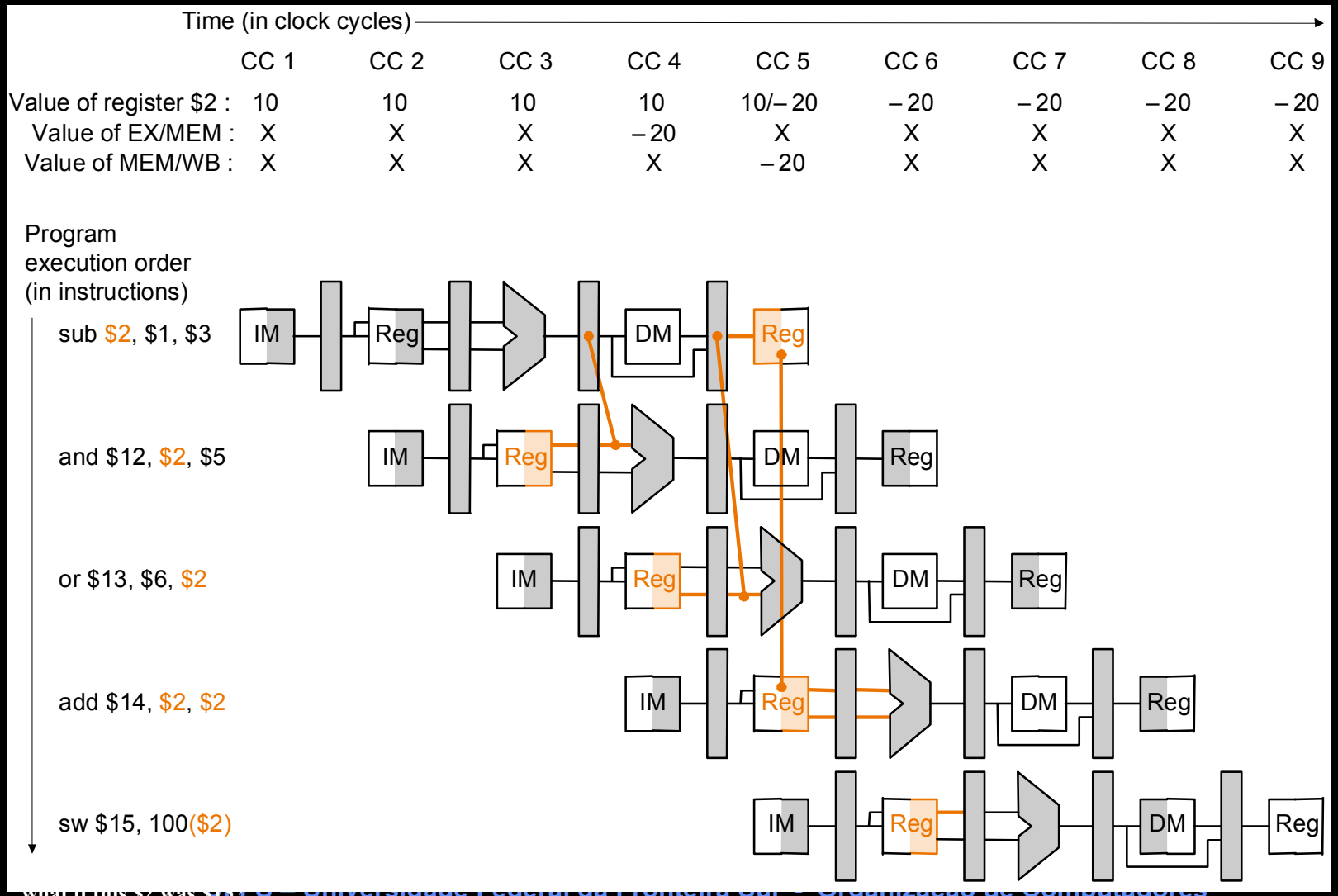
  ADD R1, R2, R3

- forwarding: caminho interno dentro do pipeline entre a saída da memória de dados e a entrada da ALU

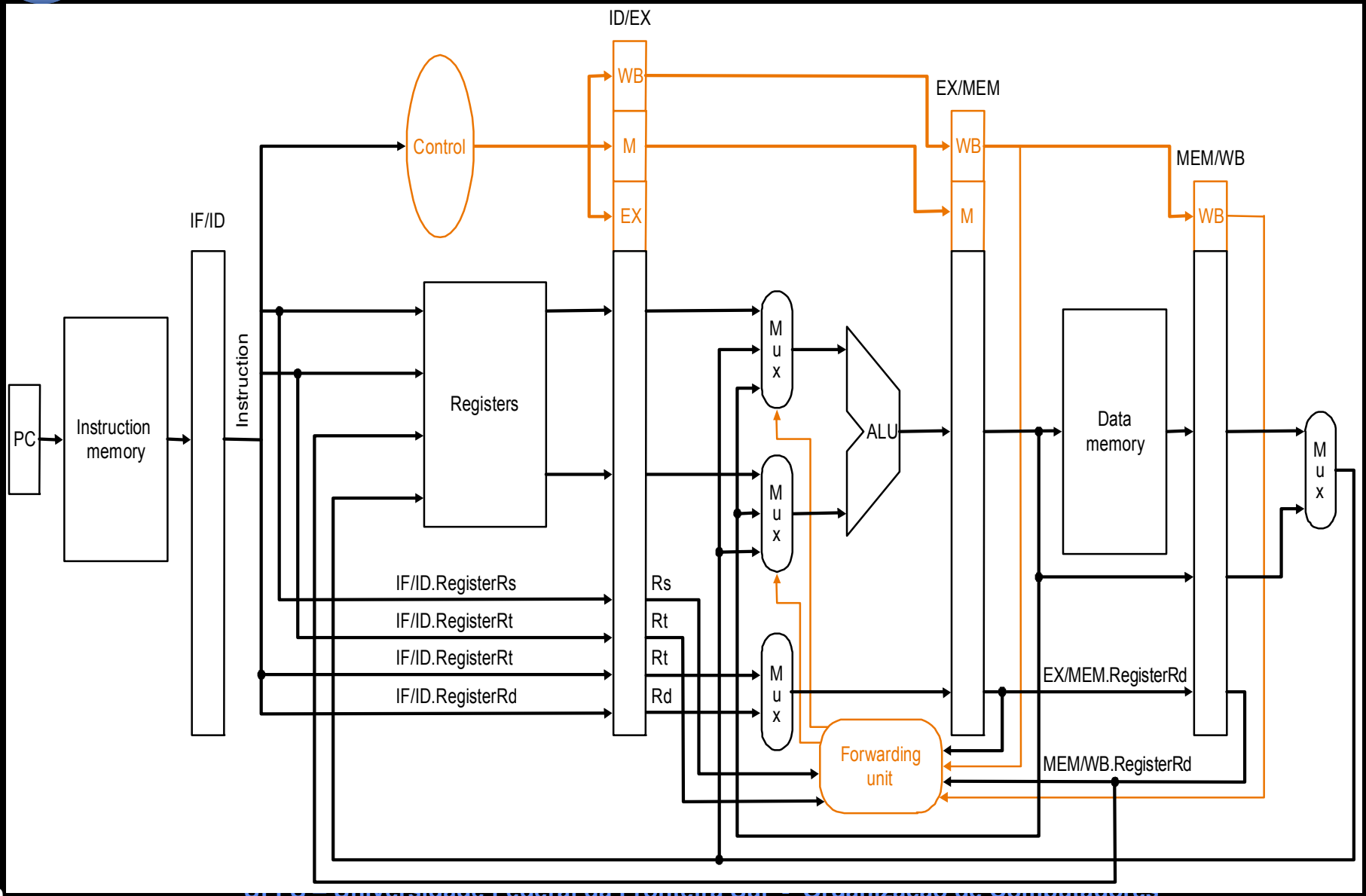


Antecipando – usar os resultados temporários, sem esperar que eles sejam escritos

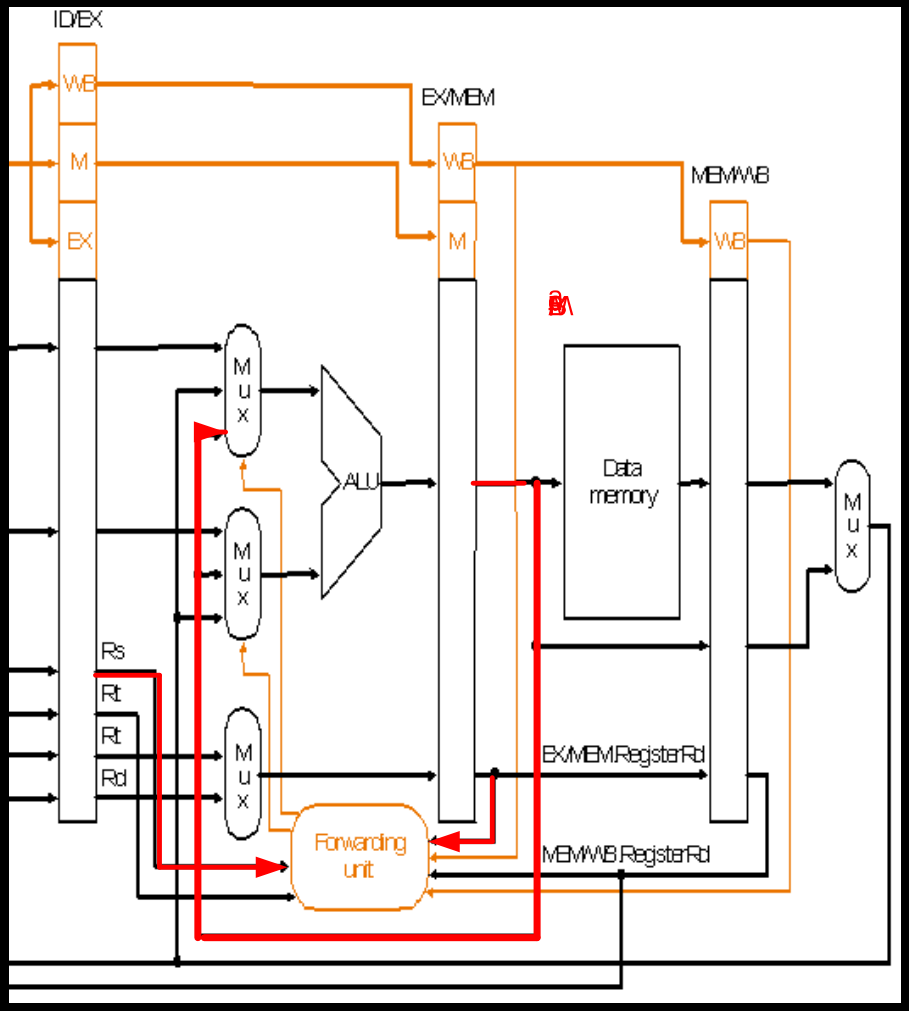
- Atuar no caminho do banco de registr. p/ substituir o valor de leit/escrita de registrador

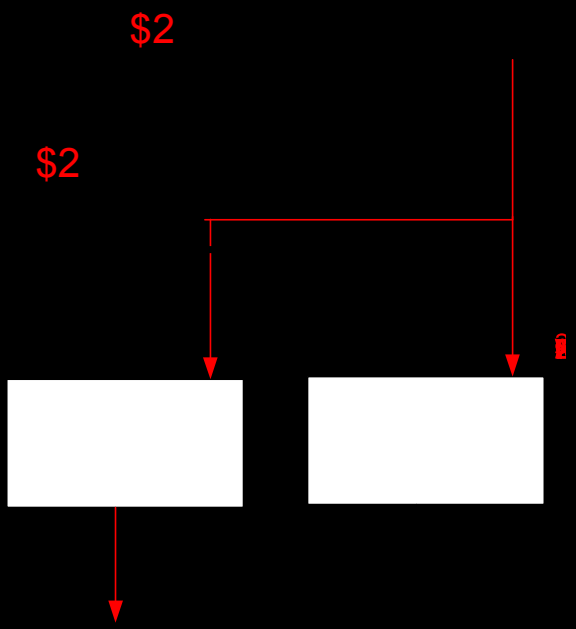


Antecipação

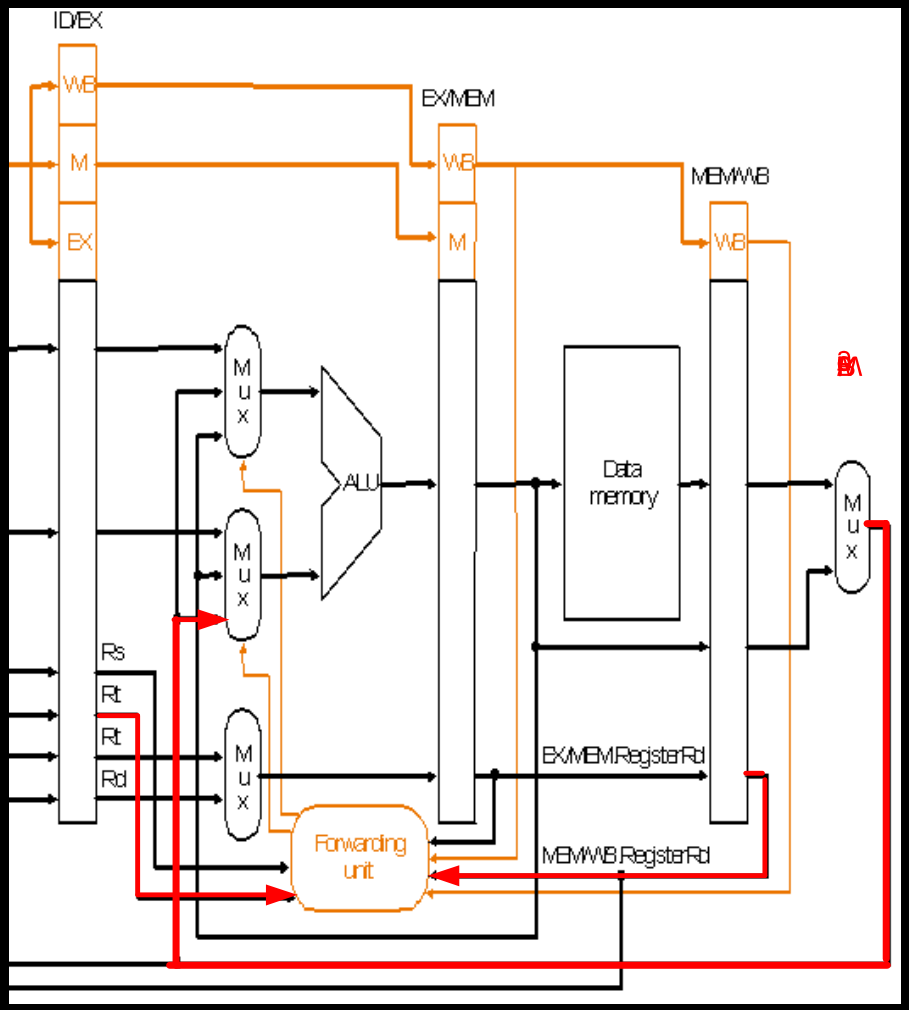


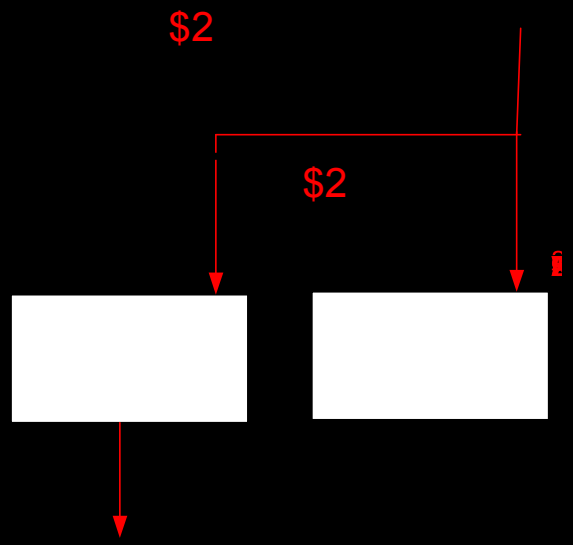
Antecipação do estágio EX/MEM





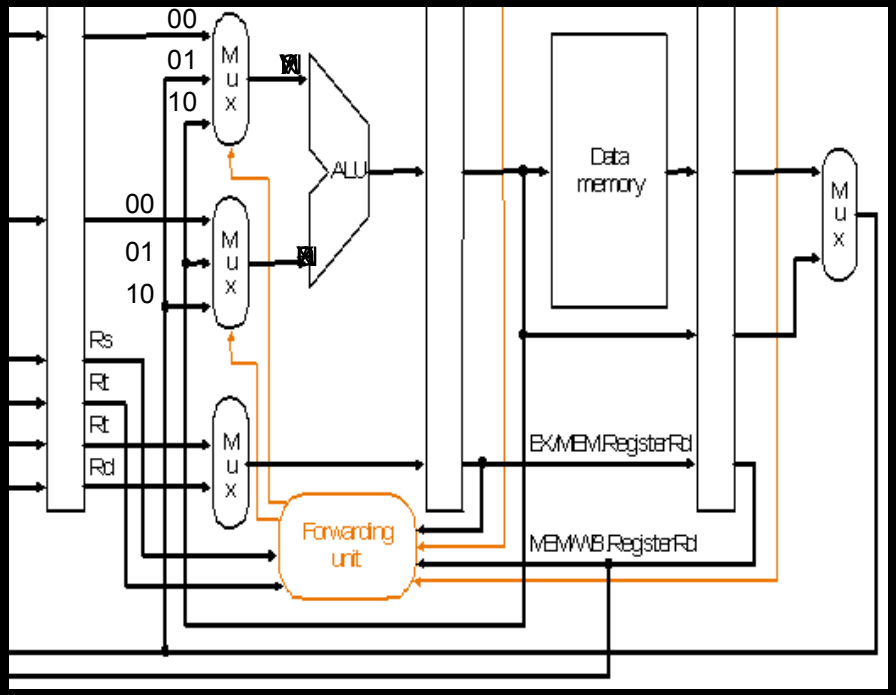
Antecipação do estágio MEM/WB

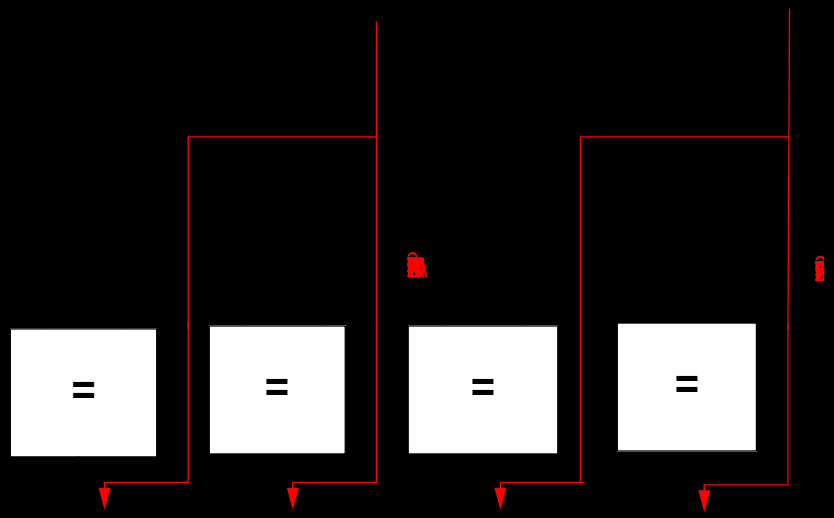




Seleciona MUX
do lado B
da ALU para
antecipação

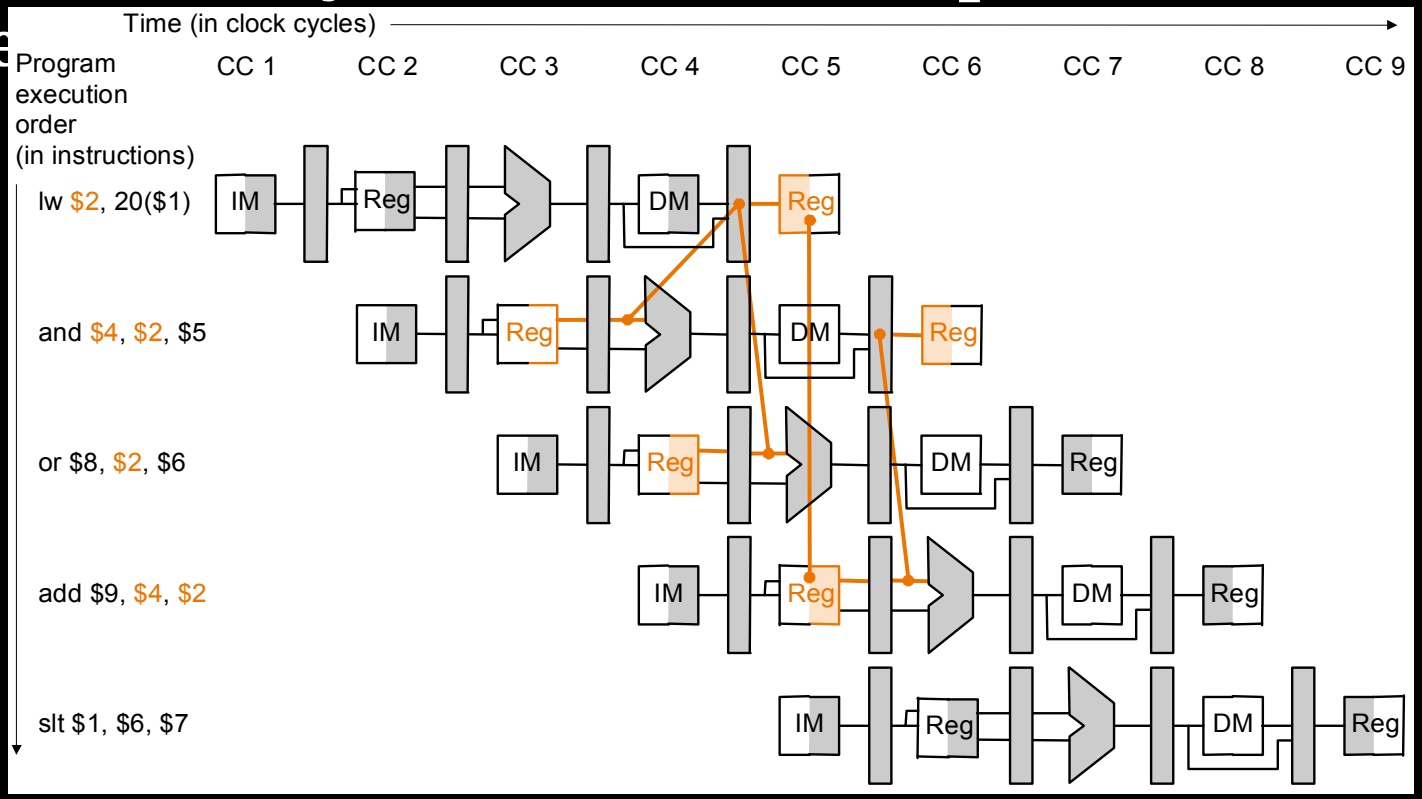
CIRCUITO DE ANTECIPAÇÃO (FORWARDING UNIT)





Nem sempre é possível (fazer o Load de uma palavra pode causar um conflito)

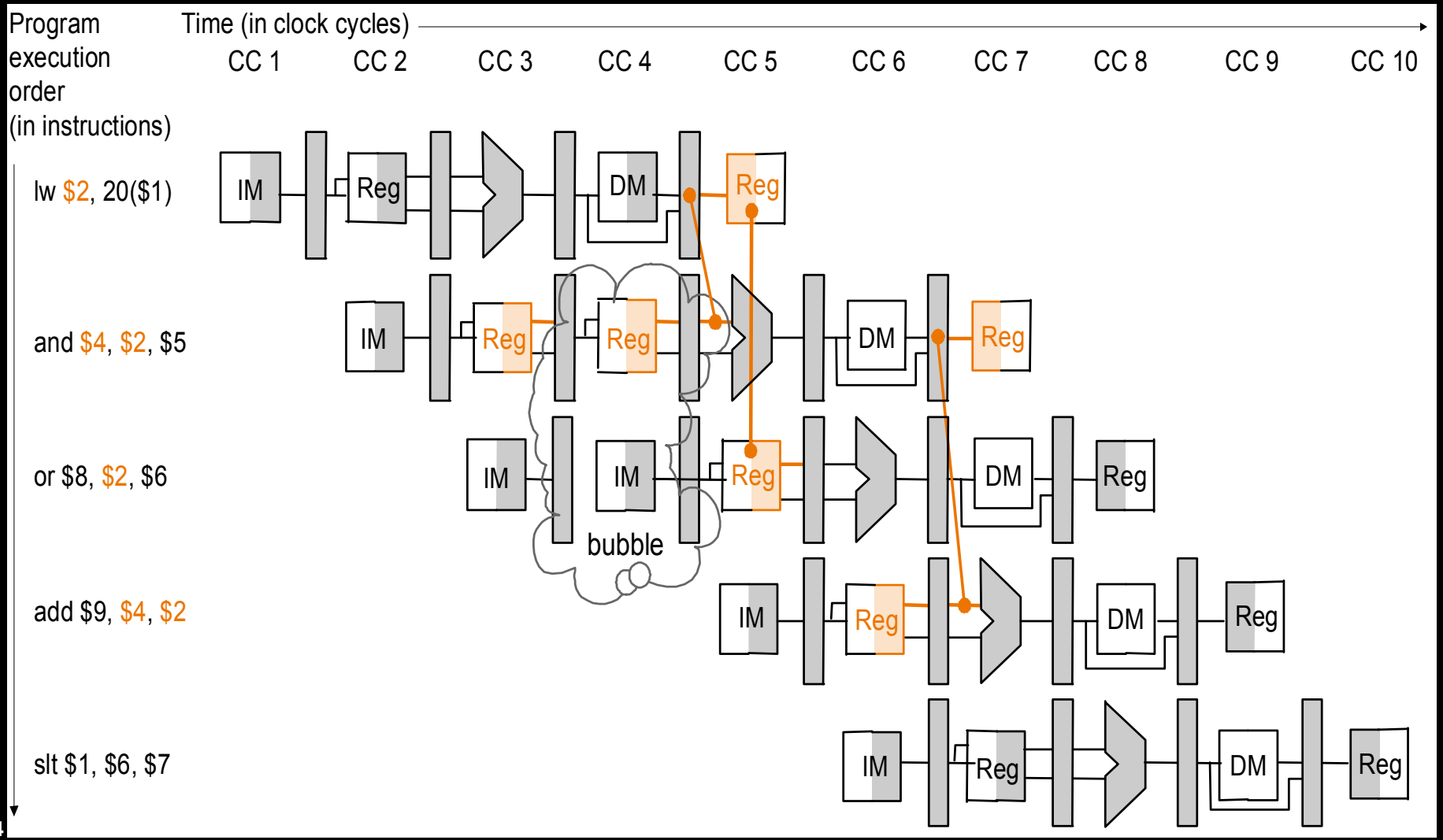
- Uma instrução tenta ler um registrador seguindo uma instrução de load word que escreve no me



- Portanto, necessitamos que a unidade de detecção de conflitos paralize, em um ciclo, as instruções seguintes ao load word

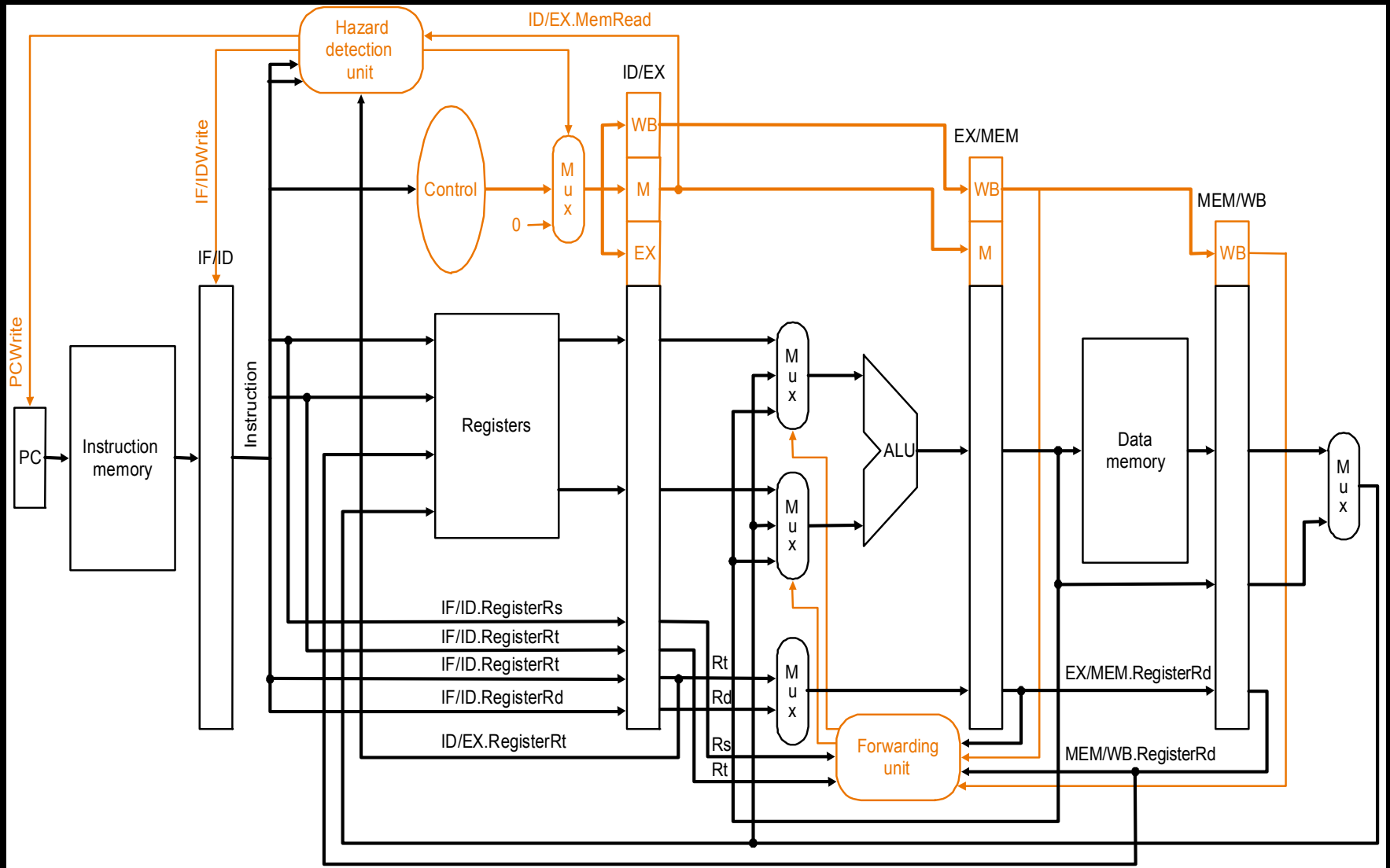
Parada

Podemos parar o pipeline mantendo uma instrução no mesmo estágio



Unidade de detecção de conflitos

A parada faz com que uma instrução que não escreve nada prossiga



Inserção de parada, com instrução NOP

or \$8,\$2,\$6 and \$4,\$2,\$5 nop ld \$2, 20(\$1)