

# **RISC-V**

## **Implementação Pipeline**

GEX 612 - Organização de Computadores

Prof. Luciano L. Caimi  
[lcaimi@uffs.edu.br](mailto:lcaimi@uffs.edu.br)

## Recursos

- Simulador Ripes



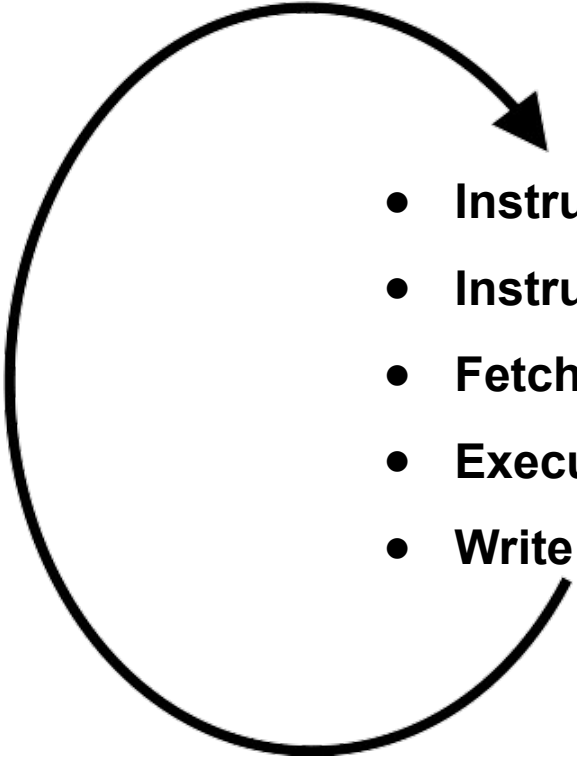
<https://github.com/mortbopet/Ripes>

- Simulador WebRISC-V

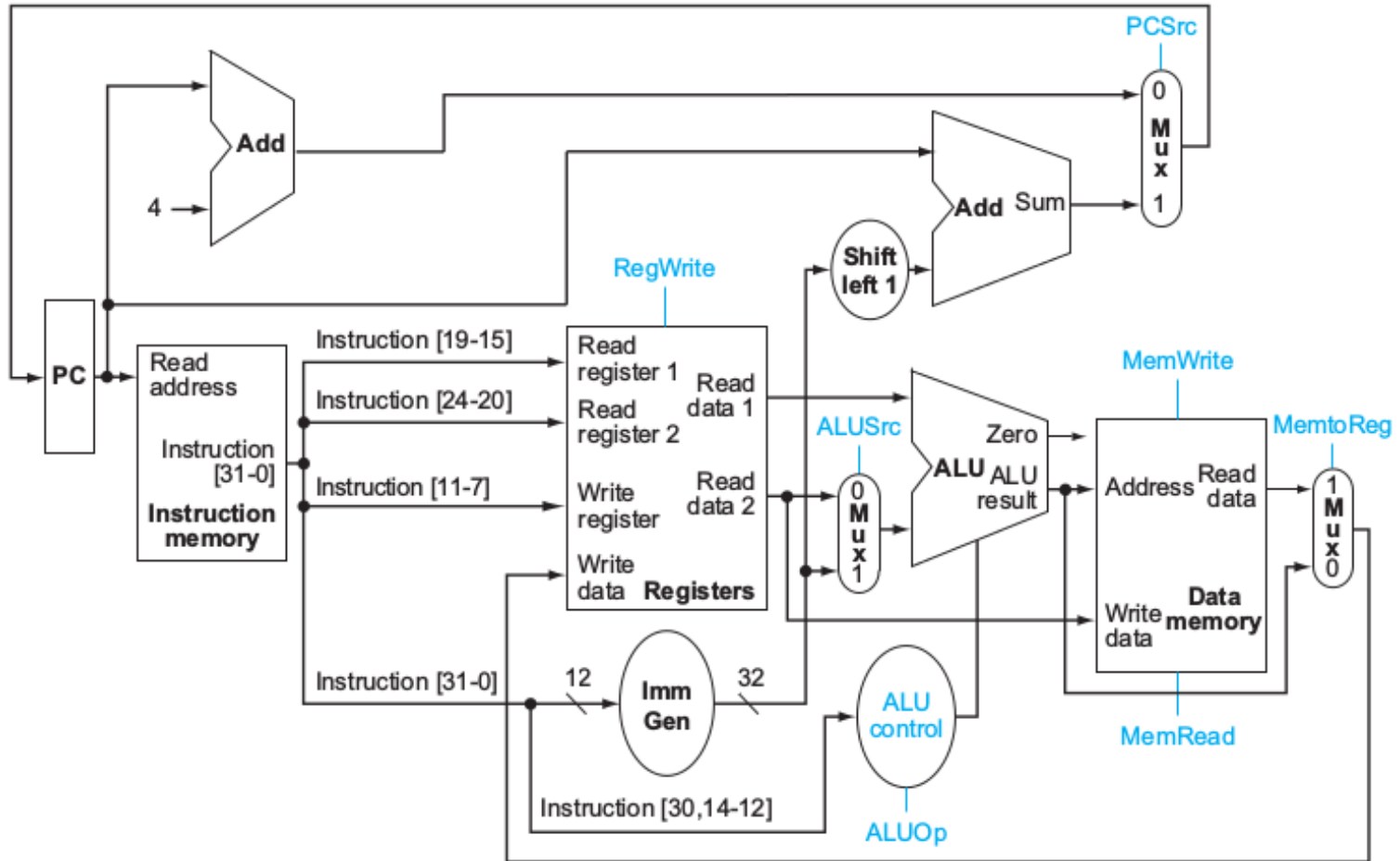
<http://x.dii.unisi.it:8098/~giorgi/WebRISC-V/index.php>



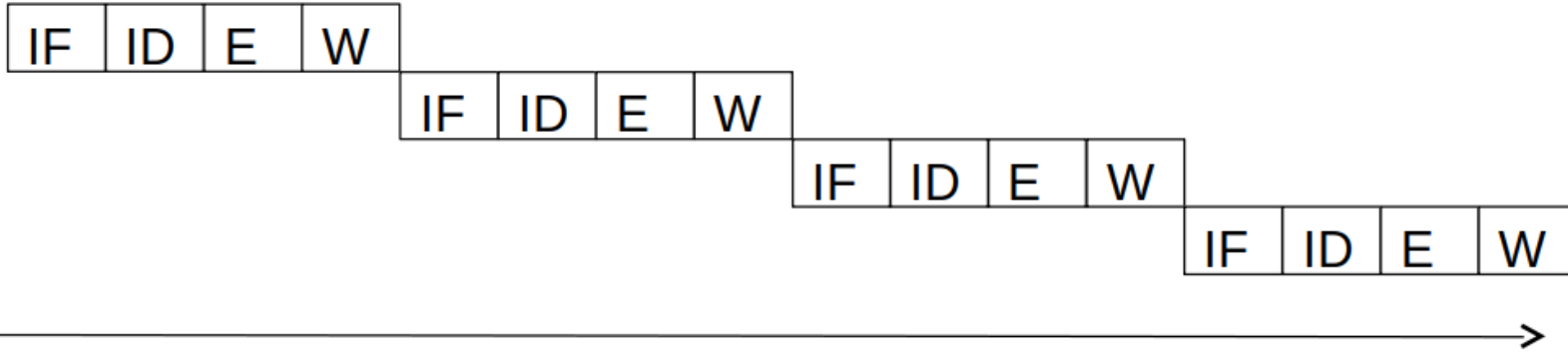
# Ciclo de Instrução

- 
- A large, thick black curved arrow starts from the bottom of the list and points back to the top, indicating a continuous cycle.
- **Instruction Fetch - IF (Busca Instrução - BI)**
  - **Instruction Decode - ID (Decodifica Instrução- DI)**
  - **Fetch Operands - FO - (Busca Operandos - BO)**
  - **Execute - EX (Executa - EX)**
  - **Write result - WB (Armazena resultado - AR)**

# Bloco operativo monociclo



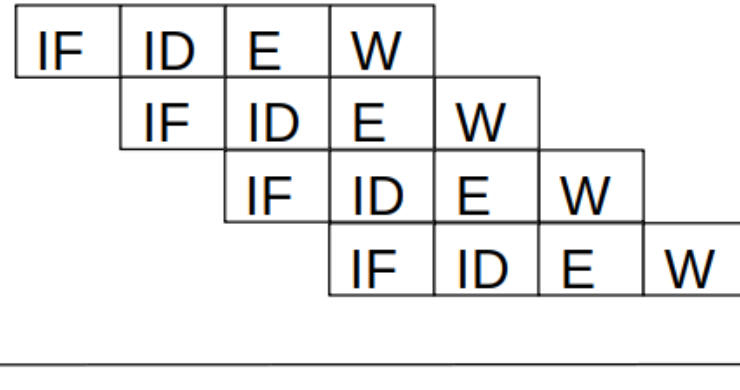
# Pipeline



- É possível otimizar a implementação?

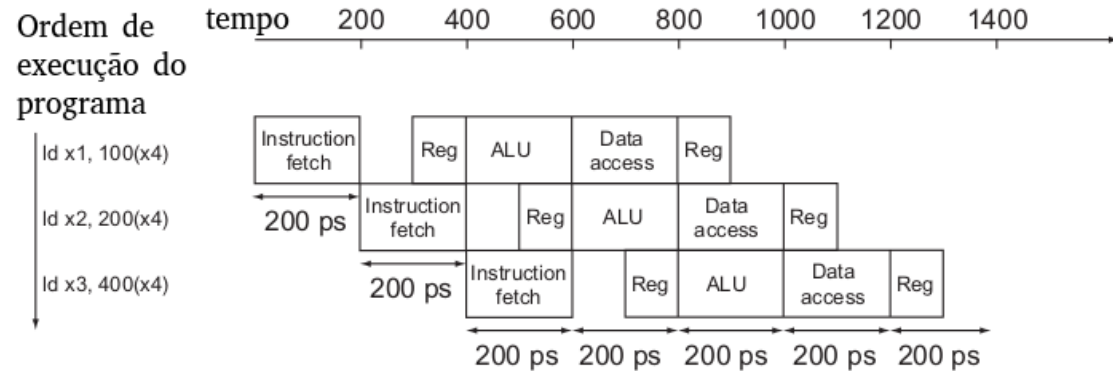
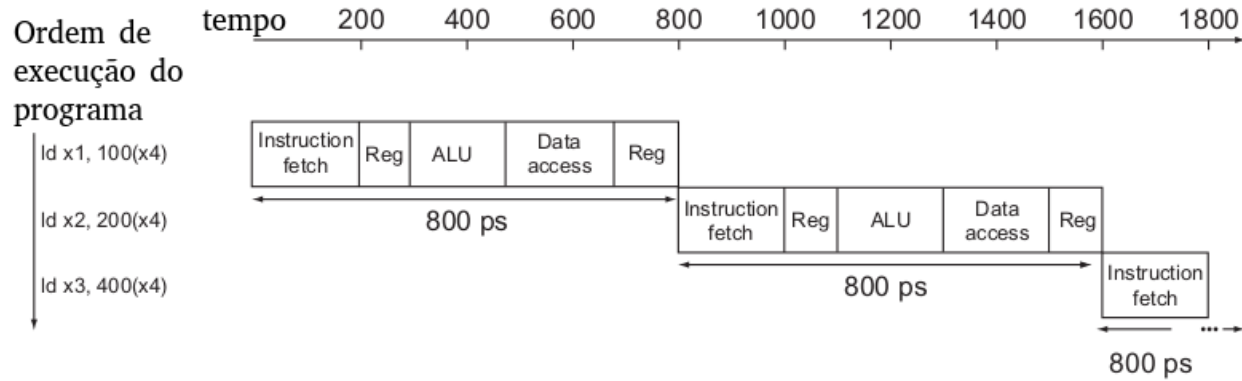
## **SIM**

Paralelismo no nível de Instrução  
(Instruction Level Parallelism - ILP)

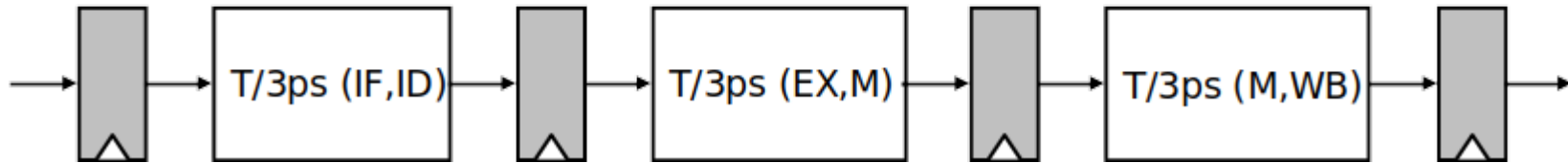
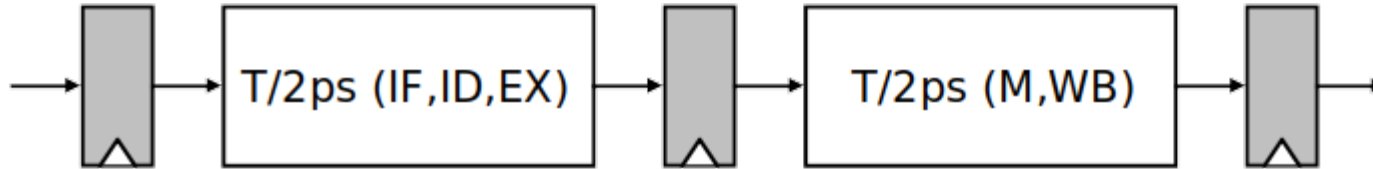
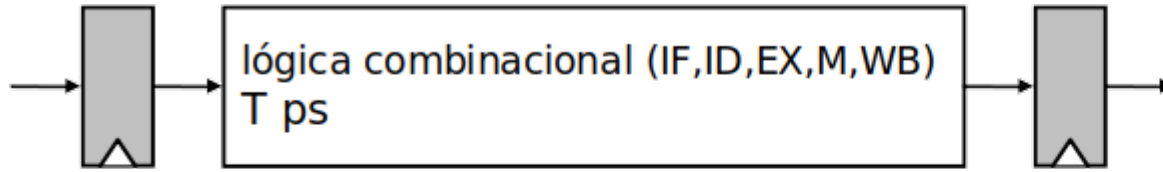


- **Objetivo:** aumento da carga de trabalho (throughput) e consequente aumento no desempenho com um baixo aumento nos custos de hardware
  - divisão de uma tarefa em N estágios
  - N instruções executadas em paralelo, uma em cada estágio
  - cada um dos N estágios particionado uniformemente
  - a mesma operação é realizada em cada estágio para diferentes instruções
- Throughput x Latência
  - Total de instruções executadas em um período de tempo
  - Tempo de execução de uma instrução

# Pipeline

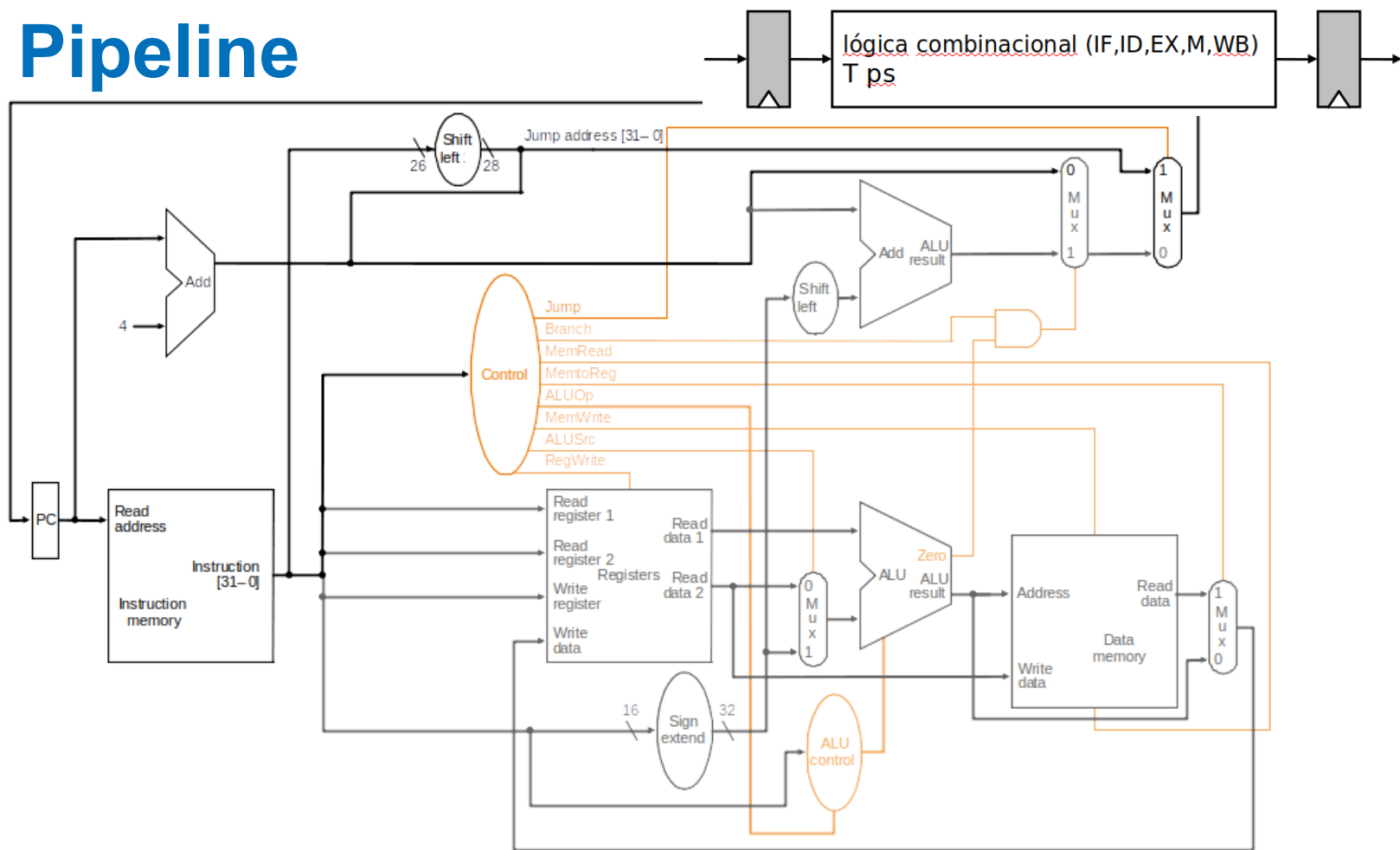


# Pipeline

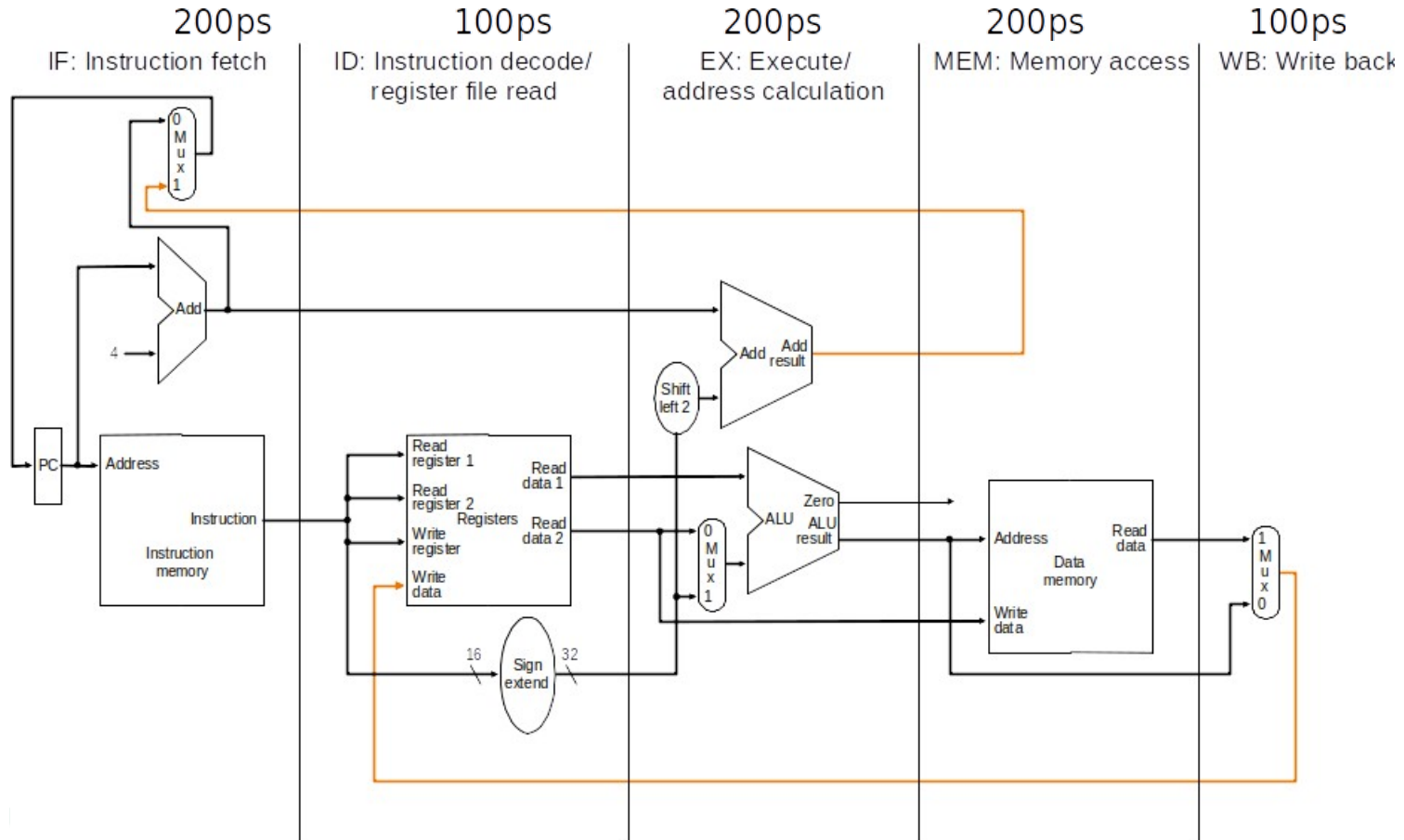




# Pipeline

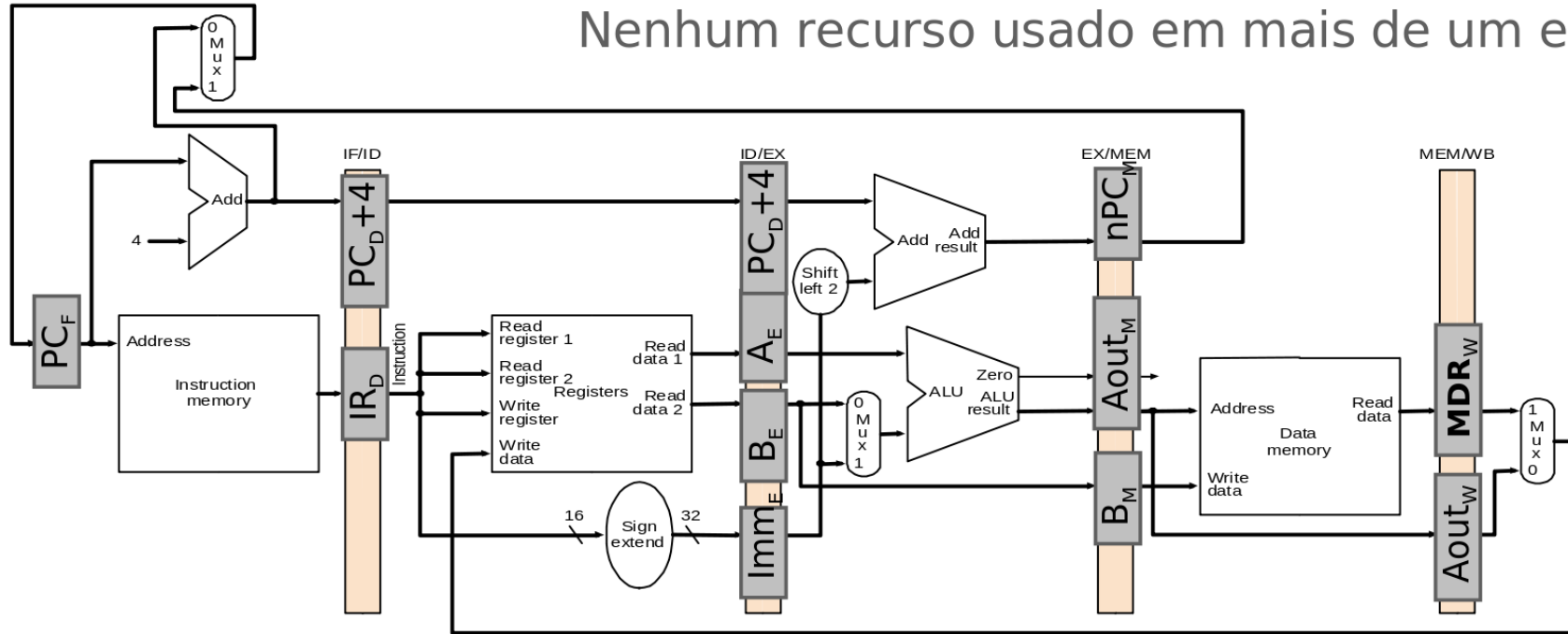


# Pipeline



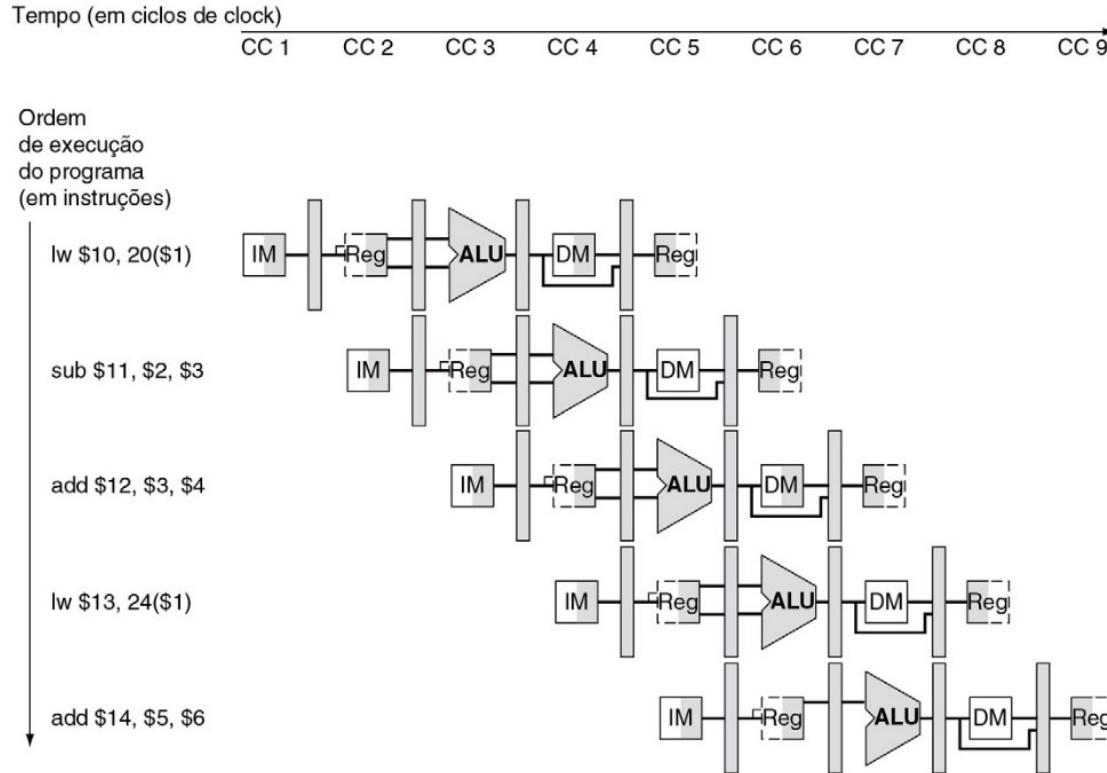
# Pipeline

Nenhum recurso usado em mais de um estágio



# Pipeline

## Representação do Pipeline:



# Pipeline

## Representação do Pipeline:

Tempo (em ciclos de clock)

CC 1 CC 2 CC 3 CC 4 CC 5 CC 6 CC 7 CC 8 CC 9

Ordem  
de execução  
do programa  
(em instruções)

lw \$10, 20(\$1)

Busca  
de instrução

Decodificação  
de instrução

Execução

Acesso  
de dados

Write-back

sub \$11, \$2, \$3

Busca  
de instrução

Decodificação  
de instrução

Execução

Acesso  
de dados

Write-back

add \$12, \$3, \$4

Busca  
de instrução

Decodificação  
de instrução

Execução

Acesso  
de dados

Write-back

lw \$13, 24(\$1)

Busca  
de instrução

Decodificação  
de instrução

Execução

Data  
access

Write-back

add \$14, \$5, \$6

Busca  
de instrução

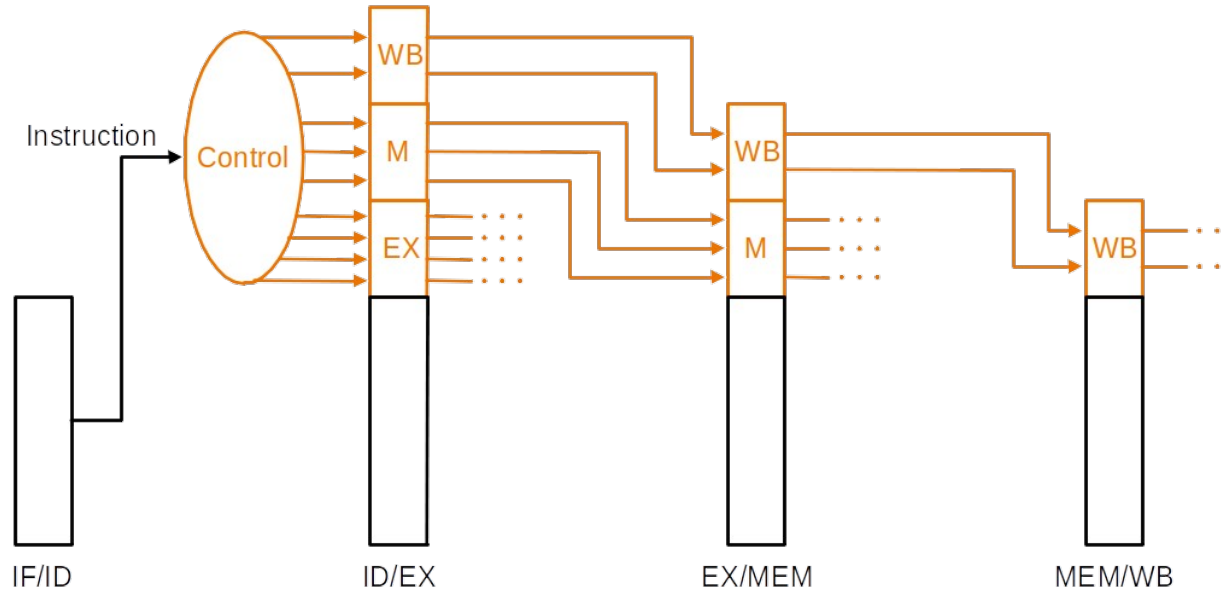
Decodificação  
de instrução

Execution

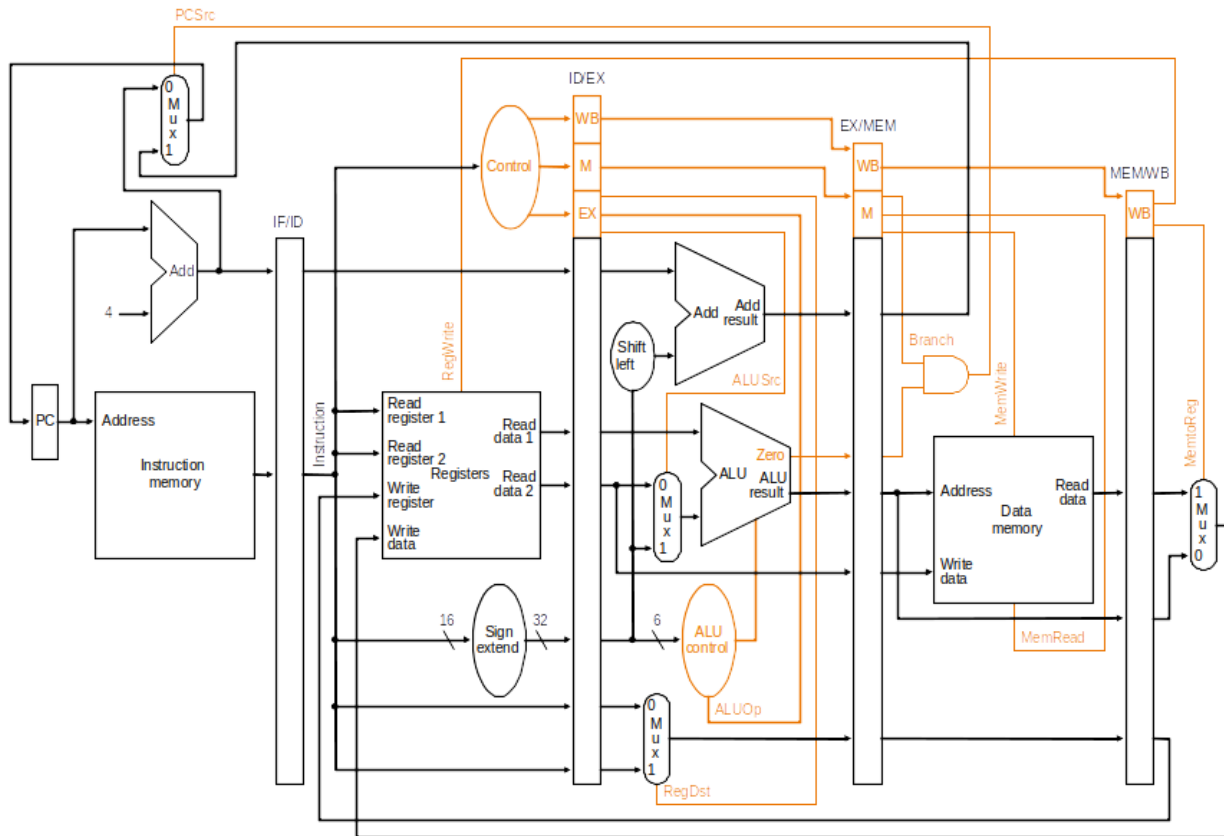
Data  
access

Write-back

# Controle do Pipeline



# Controle do Pipeline



- **Objetivo:** aumento da carga de trabalho (throughput) e consequente aumento no desempenho com um baixo aumento nos custos de hardware
- Repetição de instruções idênticas (**Não acontece**)
  - Diferentes instruções executam utilizando diferentes estágios levando a ociosidade dentro do Pipeline (exemplo LW/SW - R format) - fragmentação externa
- Cada um dos N estágios particionado uniformemente (**Não acontece**)
  - Difícil balancear a carga de trabalho realizada em cada estágio. A um dado tempo de ciclo os estágios ficam ociosos - fragmentação interna
- Repetição de operações independentes (**Não acontece**)
  - instruções não são independentes uma das outras

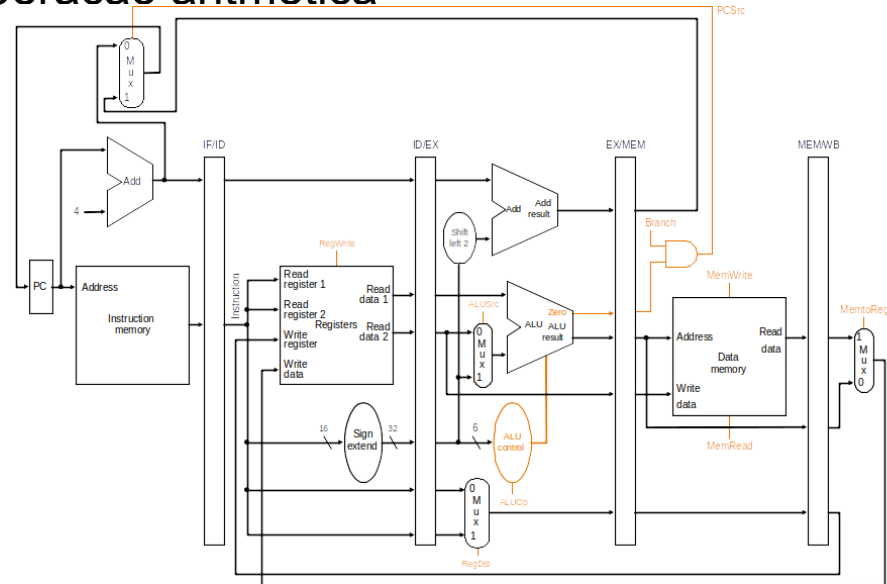


# Problemas no projeto de Pipeline



- Balanceamento da carga de trabalho entre estágios
  - Quantos estágios colocar? Quais tarefas colocar em cada estágio?
- Manter o Pipeline cheio e sem paradas mesmo diante da ocorrência de eventos que 'param' o fluxo do pipeline
  - dependências:
    - dados
    - controle
  - e conflitos (hazards):
    - estruturais
- Tratar a contenção de recursos (devido a conflitos estruturais)
- Tratar as instruções com latência grande (muitos ciclos)
- Manipular interrupções de HW

- UFFS - Universidade Federal da Fronteira Sul - Organização de Computadores 18



# Causas das paradas no Pipeline



O que acontece quando dois estágios do pipeline necessitam o mesmo recurso?

- **Solução 1:** Eliminar a causa da contenção
  - Duplicar os recursos e eliminar a causa da contenção
- **Solução 2:** Detectar a contenção e inserir uma parada (stall) no pipeline
  - Em qual estágio inserir a parada?

# Causas das paradas no Pipeline



- Dependências ou conflitos (hazards) determinadas pela ordem de execução das instruções
- Dependência entre as instruções:
  - Dependências de dados
  - Dependências de controle

# Pipeline: dependências de dados



- Tipos de dependências de dados:
  - Dependência de fluxo - dependência verdadeira: read after write (RAW)
  - Dependências de saída: write after write (WAW)
  - Anti dependência: write after read (WAR)

Dependência verdadeira

$r_3 \leftarrow r_1 \text{ op } r_2$   
 $r_5 \leftarrow r_3 \text{ op } r_4$

Read-after-Write  
(RAW)

Anti dependência

$r_3 \leftarrow r_1 \text{ op } r_2$   
 $r_1 \leftarrow r_4 \text{ op } r_5$

Write-after-Read  
(WAR)

Dependência de saída

$r_3 \leftarrow r_1 \text{ op } r_2$   
 $r_5 \leftarrow r_3 \text{ op } r_4$   
 $r_3 \leftarrow r_6 \text{ op } r_7$

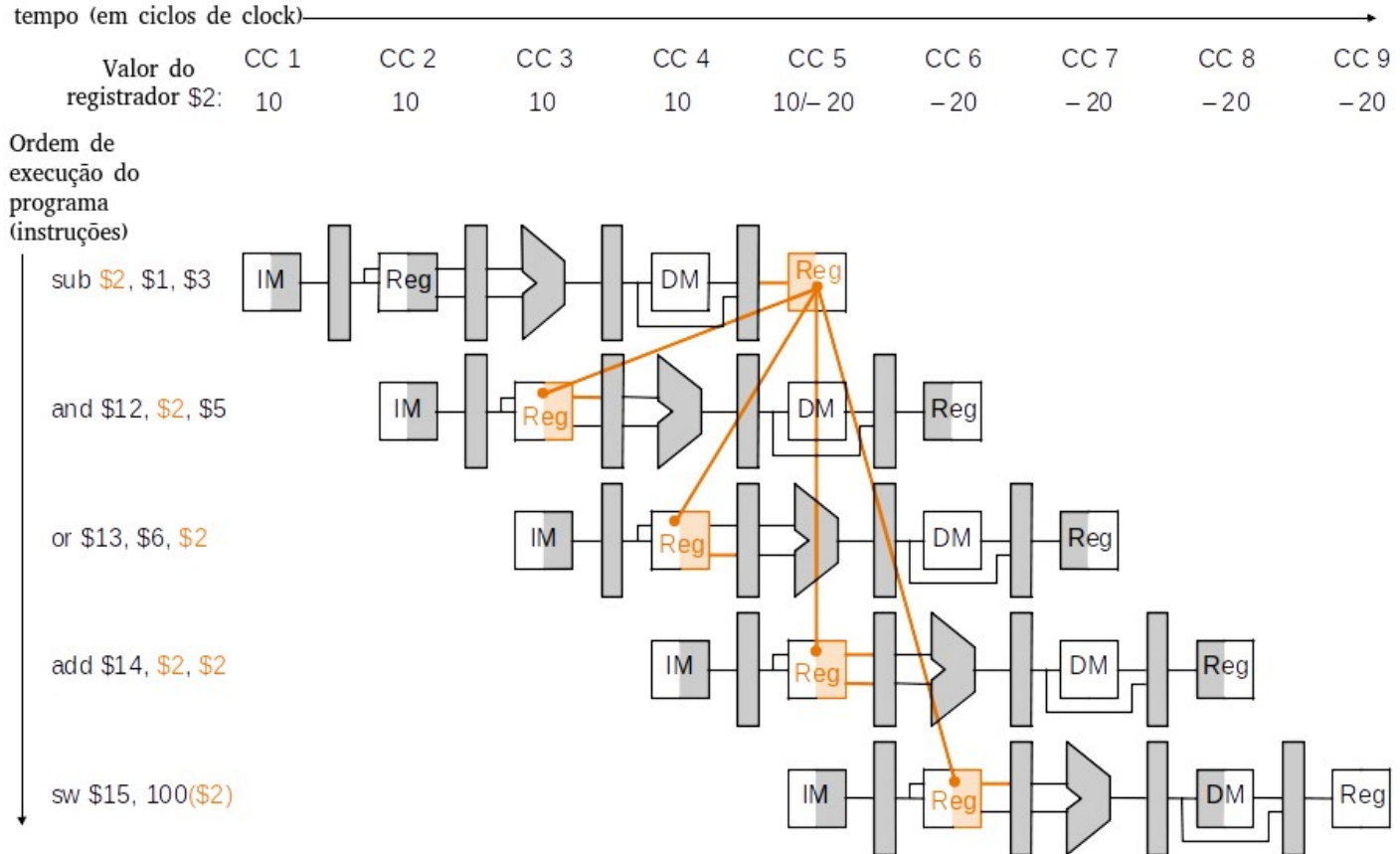
Write-after-Write  
(WAW)

# Pipeline: dependências de dados



- Tipos de dependências de dados:
  - Dependência de fluxo - dependência verdadeira: read after write (RAW)
  - Dependências de saída: write after write (WAW)
  - Anti dependência: write after read (WAR)
- Apenas a dependência verdadeira afeta o funcionamento do pipeline (as demais ocorrem em arquiteturas superescalares)
- Anti dependência e dependência de saída ocorrem devido ao número limitado de registradores (solucionado com uma técnica de renomeação de registradores)

# Pipeline: dependências de dados



# Pipeline: dependências de dados

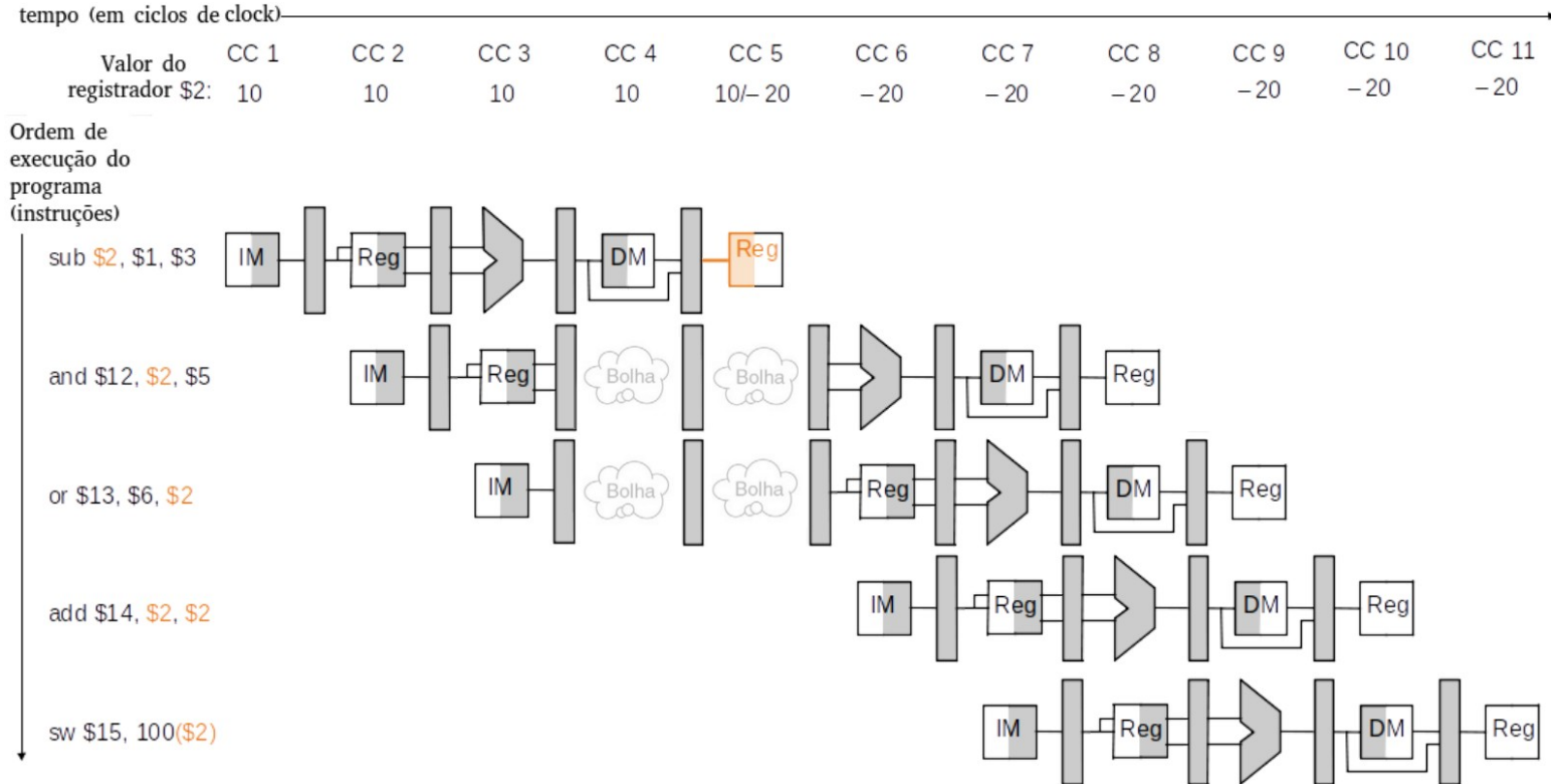


- Como tratar as dependências verdadeiras:
  - 1) detectar e esperar o registrador ficar disponível
    - inserir bolhas (stalls)
  - 2) detectar e eliminar a dependência no nível de software
    - compilador se encarrega de 'preencher' slots vazios
  - 3) detectar e adiantar o dado para a instrução dependente
    - realizar forwarding
  - 4) prever a necessidade do valor e executar de forma especulativa verificando se acertou a previsão
    - execução especulativa



# Pipeline: dependências de dados

## 1) Detectar e esperar o registrador ficar disponível



# Pipeline: dependências de dados



1) Detectar e esperar o registrador ficar disponível

Como detectar?

- Deve inserir a bolha quanto a instrução no estágio ID deseja ler um registrador que deve ser escrito e que se encontra nos estágios EX, MEM ou WB

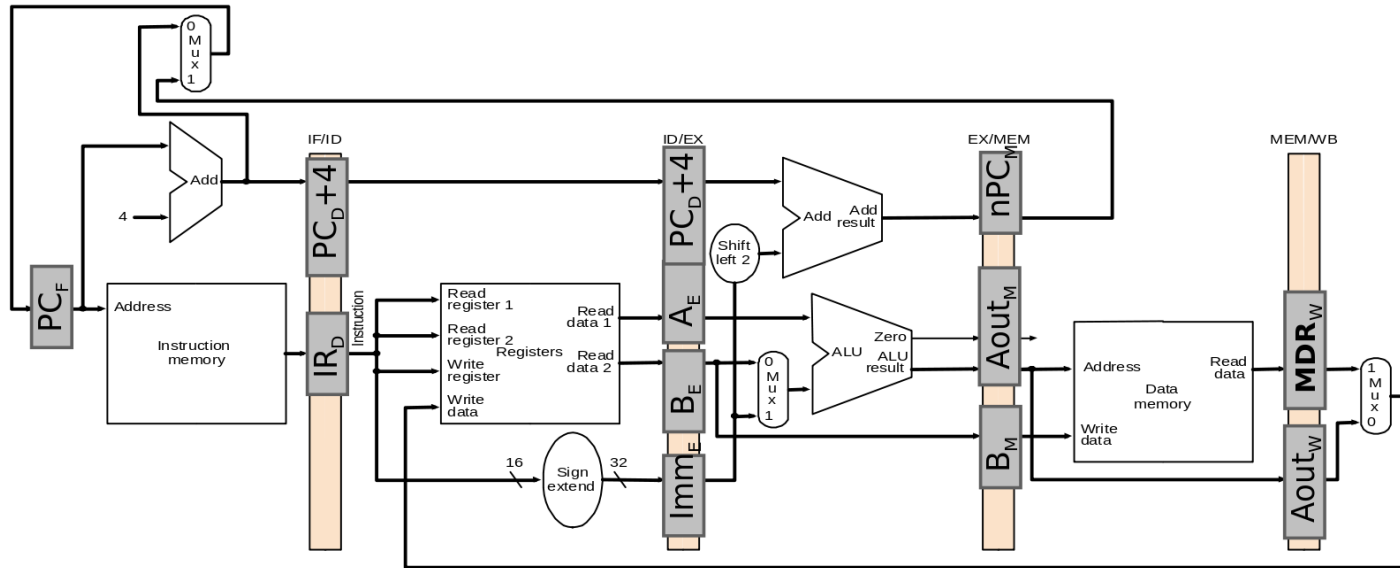
# Pipeline: dependências de dados

1) Detectar e esperar o registrador ficar disponível

Como resolver?

Estágios anteriores:

- Desabilitar escrita em PC, IR e PC+4



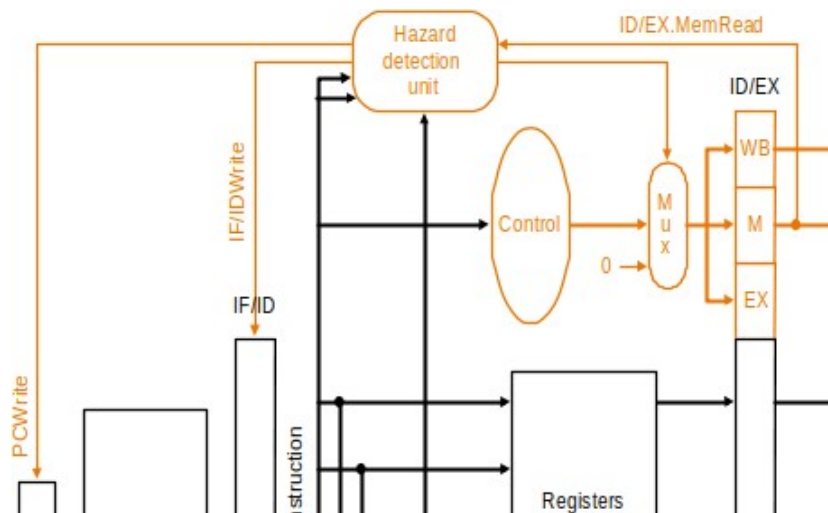
# Pipeline: dependências de dados

1) Detectar e esperar o registrador ficar disponível

Como resolver?

Estágios posteriores:

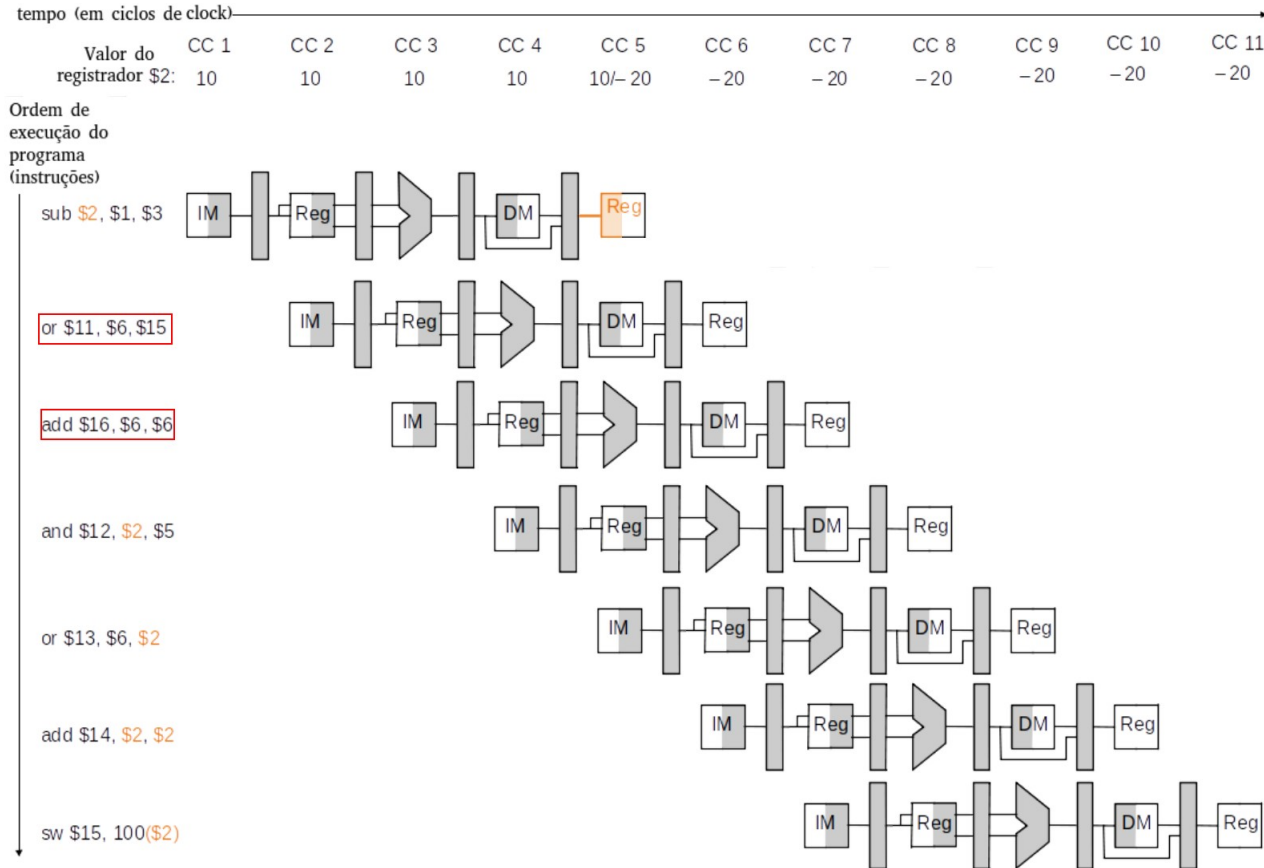
- Insere instrução NOP nos estágio seguinte até resolver a dependência  
add \$zero, \$zero, \$zero





# Pipeline: dependências de dados

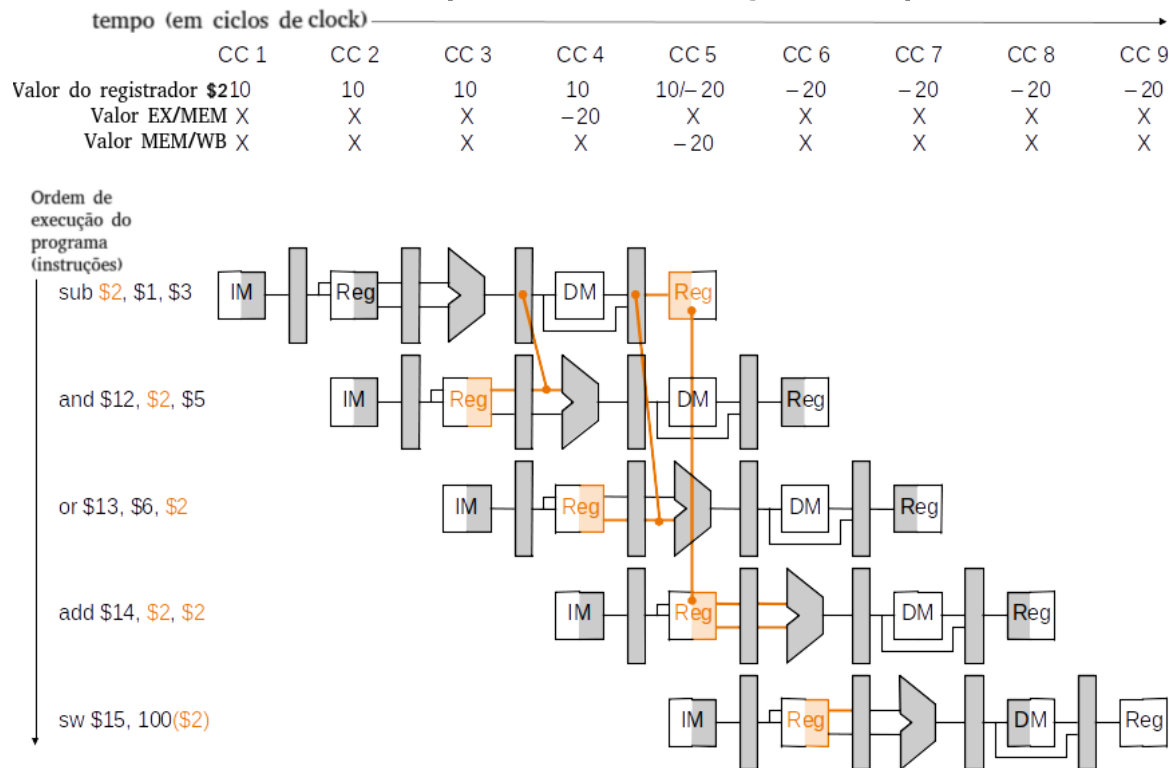
## 2) Detectar e eliminar a dependência no nível de software





# Pipeline: dependências de dados

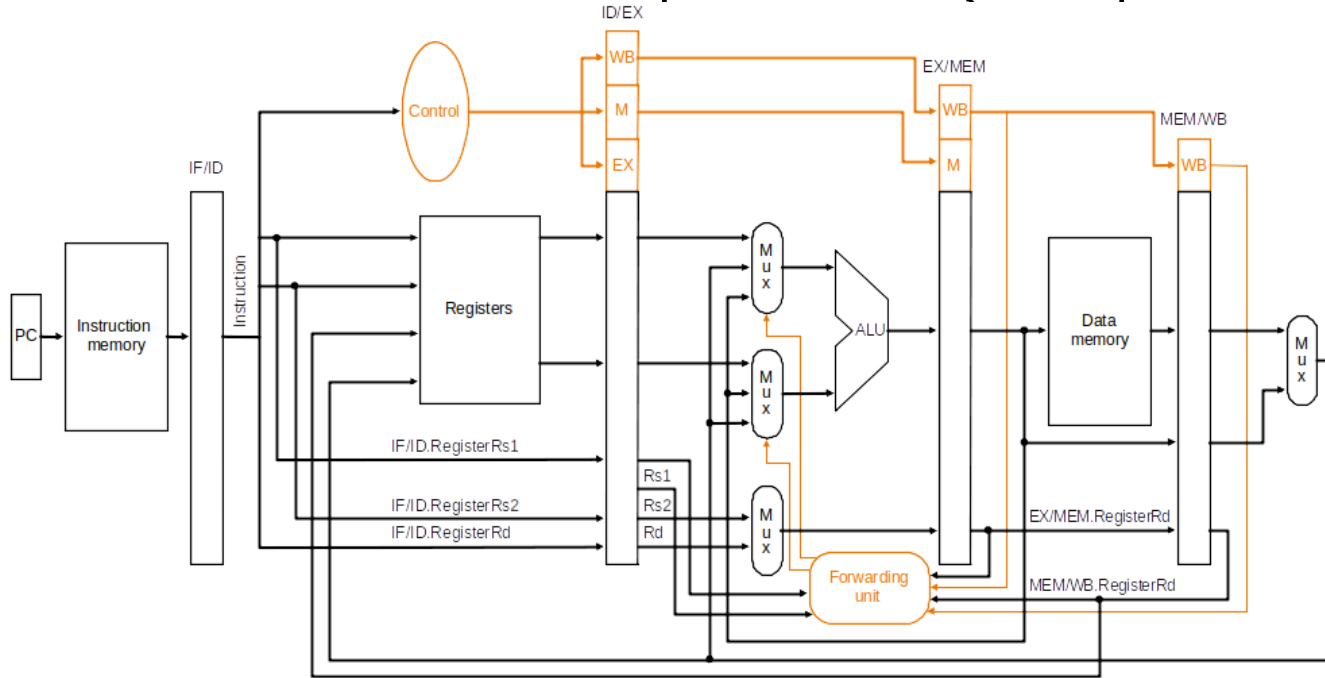
## 3) Detectar e adiantar o dado para a instrução dependente





# Pipeline: dependências de dados

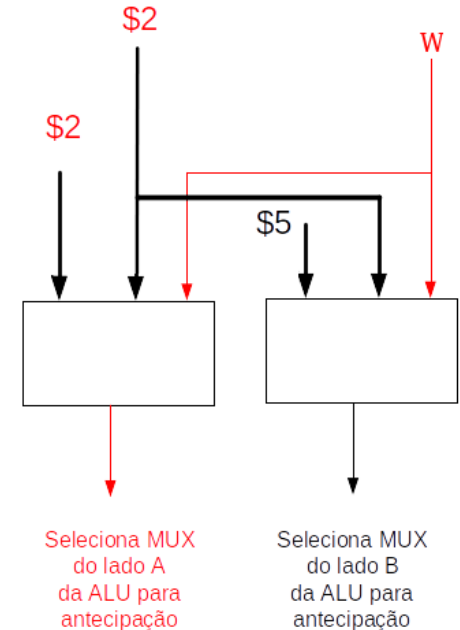
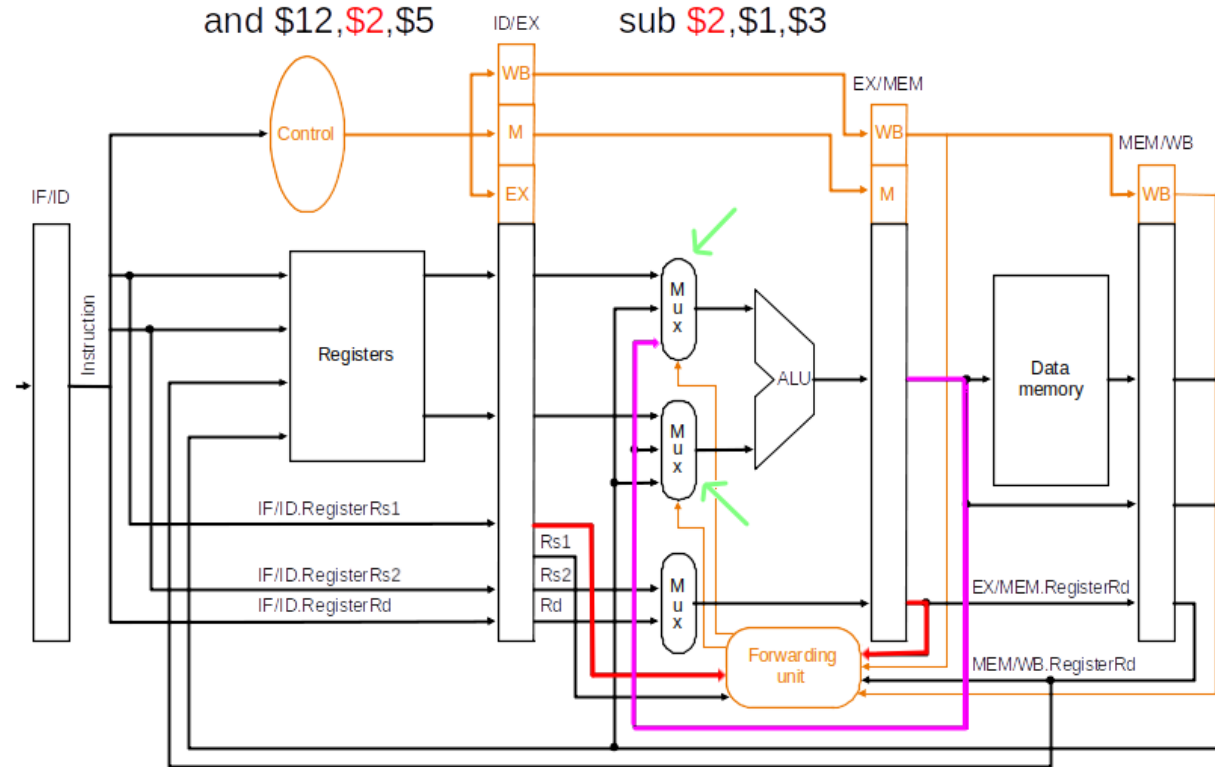
## 3) Detectar e adiantar o dado para a instrução dependente





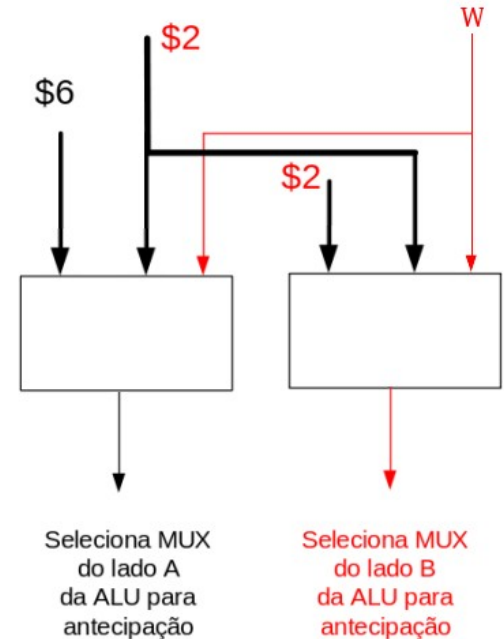
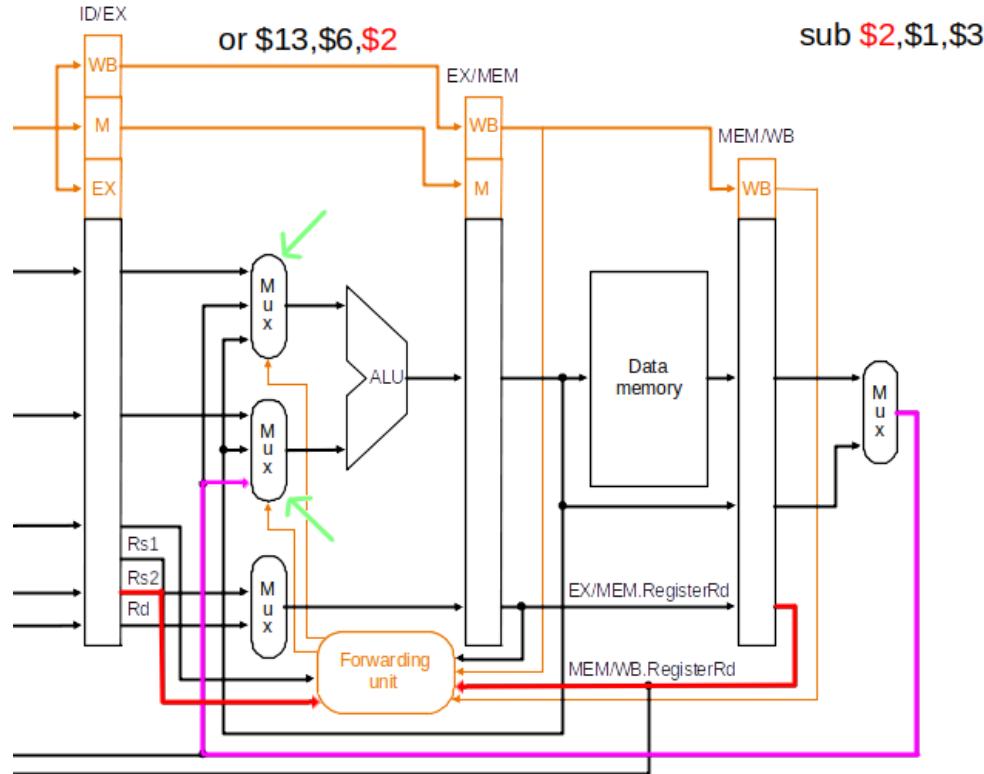
# Pipeline: dependências de dados

## 3) Detectar e adiantar o dado para a instrução dependente



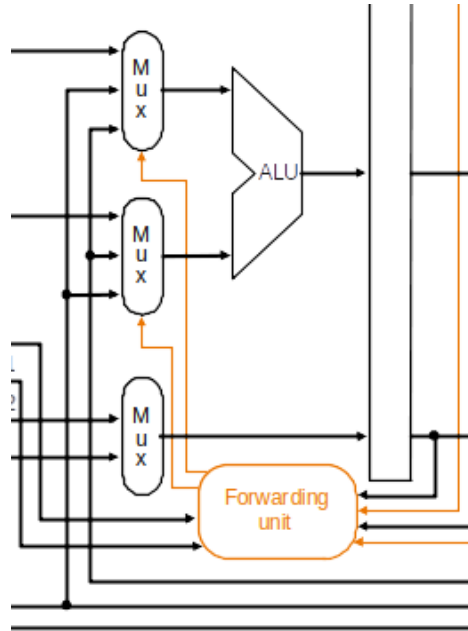
# Pipeline: dependências de dados

## 3) Detectar e adiantar o dado para a instrução dependente

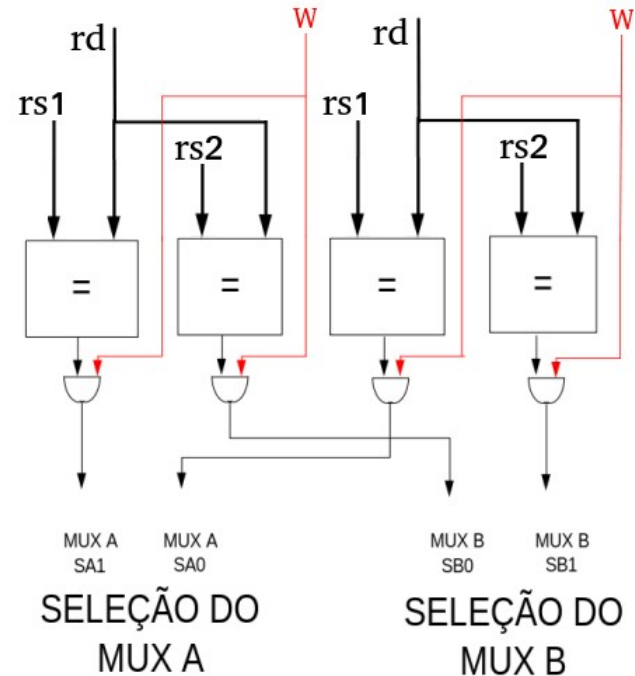


# Pipeline: dependências de dados

## 3) Detectar e adiantar o dado para a instrução dependente

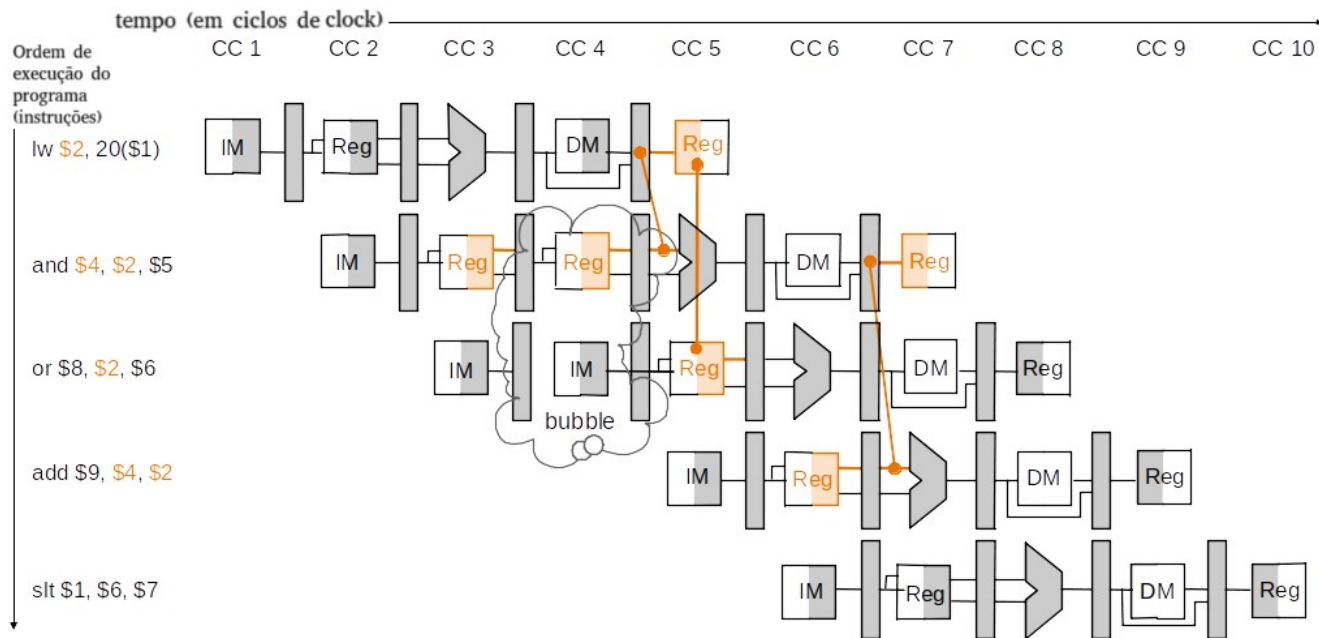


(estágio EX/MEM) (estágio MEM/WB)



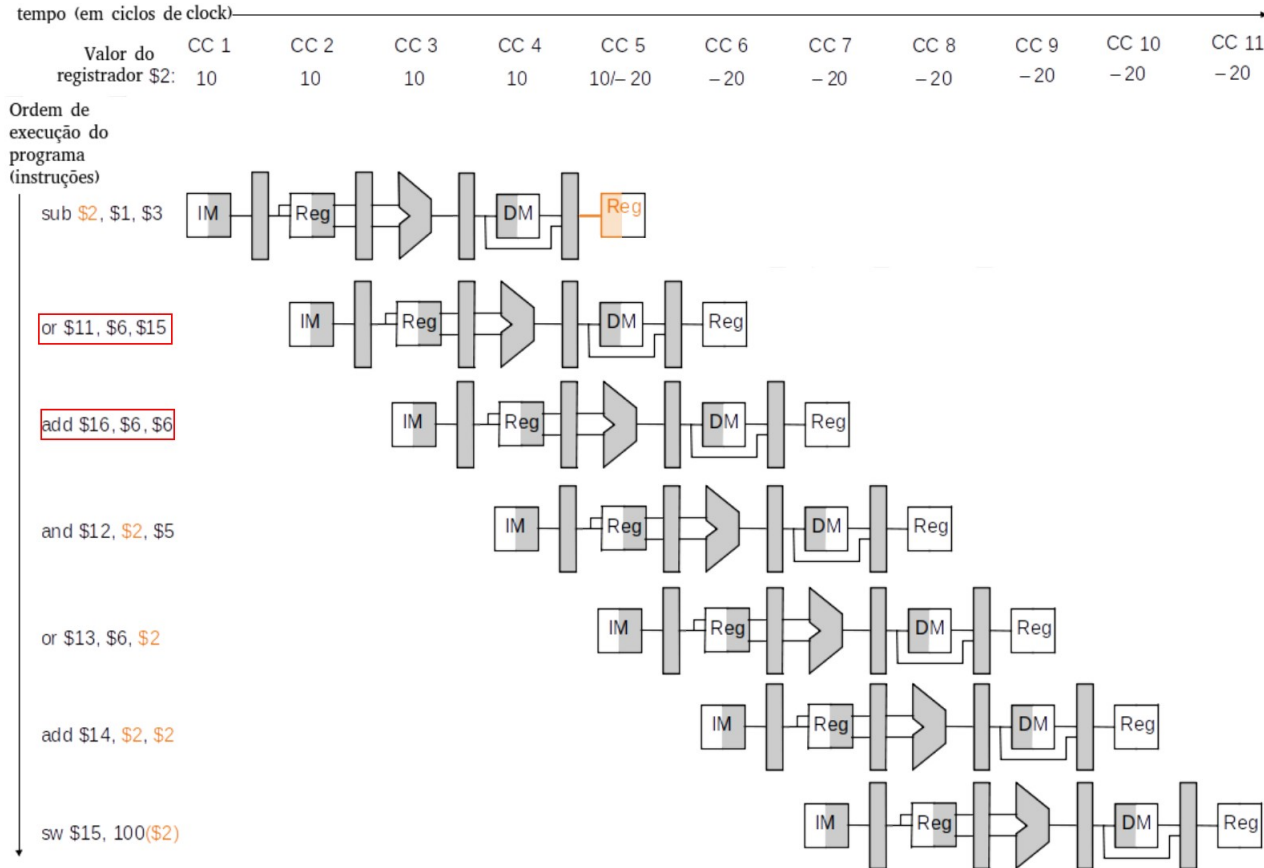
# Pipeline: dependências de dados

- 3) Detectar e adiantar o dado para a instrução dependente  
Mesmo fazendo adiantamento (forwarding) as vezes é preciso inserir bolha (stall)



# Pipeline: dependências de dados

## 4) Detectar e eliminar a dependência no nível de software



# Pipeline: dependências de controle



- **Pergunta:** Qual deve ser a próxima instrução a ser buscada?

**Resposta:** A instrução que estiver no endereço indicado pelo PC

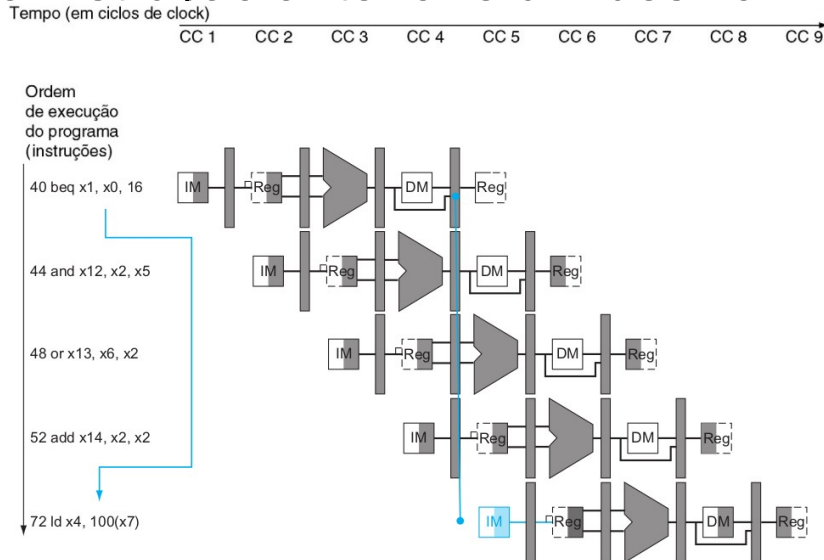
- Todas as instruções a serem executadas, em algum sentido, dependem da instrução anterior
  - Se a instrução estiver no endereço de memória seguinte tudo ocorre maravilhosamente bem (aritméticas, acesso a memória)
  - Se a instrução desviar o fluxo de execução do programa teremos problemas (desvios condicionais e incondicionais)
- Aspecto crítico para manter o pipeline cheio com a sequência correta de instruções



# Pipeline: dependências de controle



- As dependências de controle em um pipeline ocorrem devido a mudança do fluxo de execução das instruções
- Ocorre quando o novo valor de PC não é  $PC+4$ , ou seja, quando a instrução anterior é um desvio



# Pipeline: dependências de controle



- Diferentes tipos de desvio possuem diferentes comportamentos
- Diferentes tipos de desvio são tratados de forma distinta

Tipo	Direção no momento da busca	Número de possíveis endereços na próxima busca?	Quando o endereço da próxima busca é resolvido?
Condicional (ex: beq)	Desconhecido	2	Execução (dependente do registrador)
Incondicional (ex: j)	Sempre tomado (Always taken)	1	Decodificação (PC + offset)
Call	Sempre tomado (Always taken)	1	Decodificação (PC + offset)
Return	Sempre tomado (Always taken)	Muitos	Execução (dependente do registrador)
Indireto	Sempre tomado (Always taken)	Muitos	Execução (dependente do registrador)

# Pipeline: dependências de controle

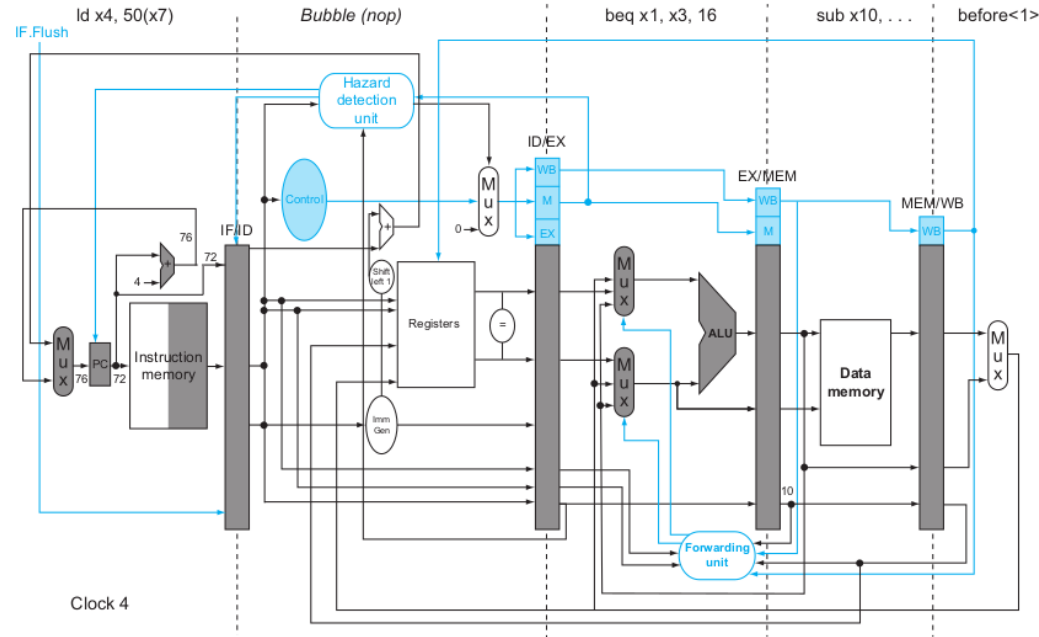


- Possíveis soluções para manter o comportamento funcional do programa correto na presença de desvios no pipeline:
  - a. Inserir bolhas (stalls) no pipeline até conhecer o endereço da próxima instrução a ser executada
  - b. Utilizar desvio atrasado (delayed branch)
  - c. Utilizar execução condicional (predicated execution)
  - d. Buscar os dois endereços do desvio e executar os dois fluxos de instruções possíveis (multipath execution - superescalar)
  - e. Utilizar predição de desvio (branch prediction)

# Pipeline: dependências de controle



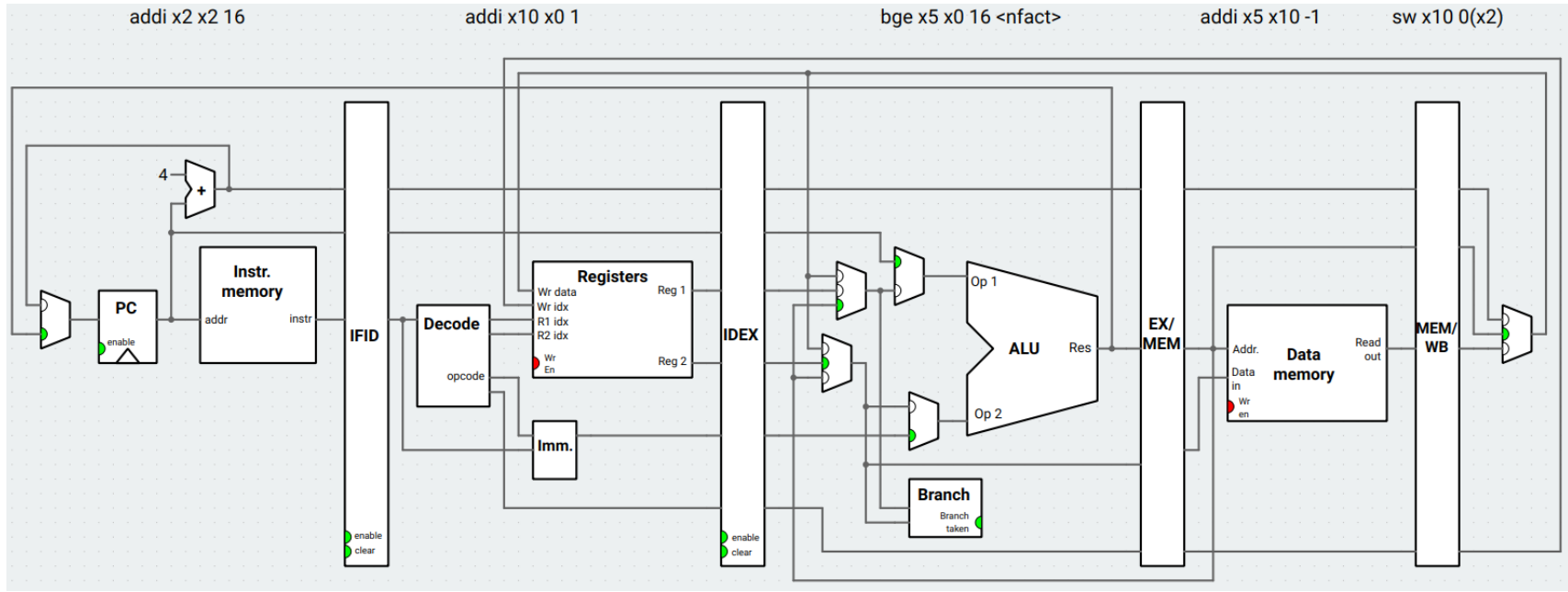
- a. Inserir bolhas (stalls) no pipeline até conhecer o endereço da próxima instrução a ser executada (flush instructions)



# Pipeline: dependências de controle



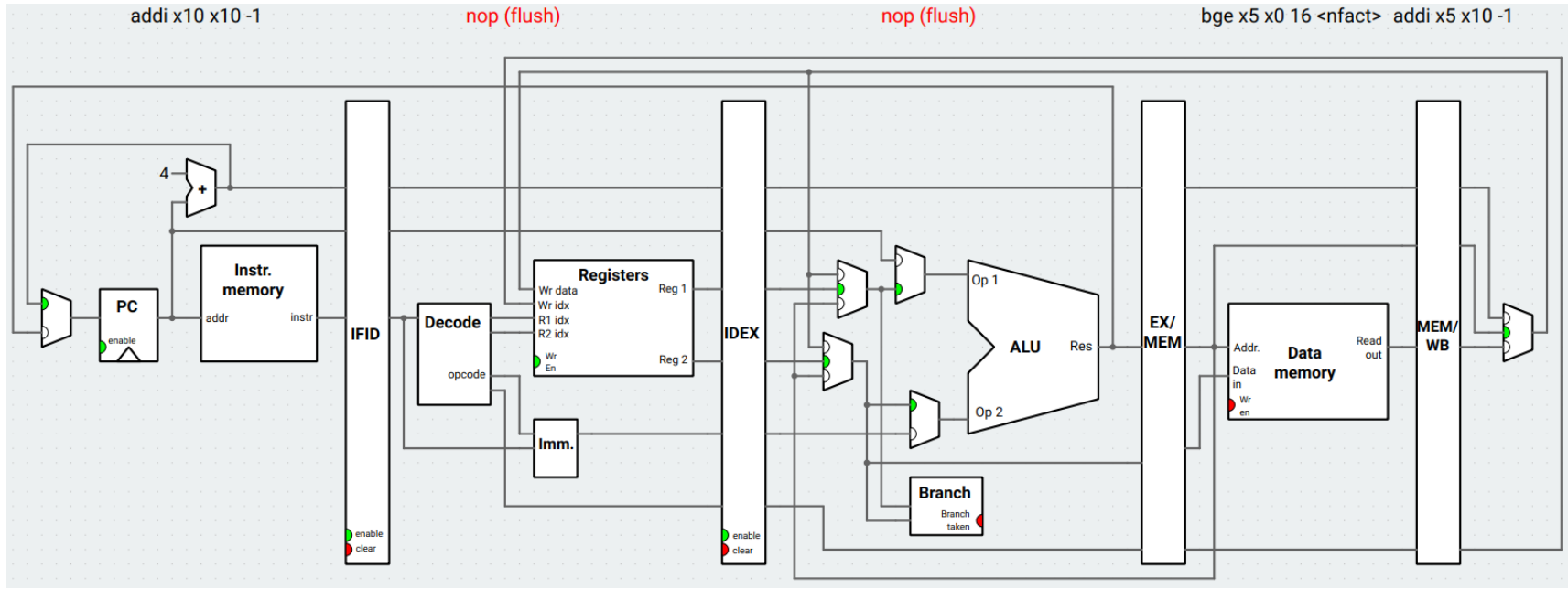
- a. Inserir bolhas (stalls) no pipeline até conhecer o endereço da próxima instrução a ser executada (flush instructions)



# Pipeline: dependências de controle

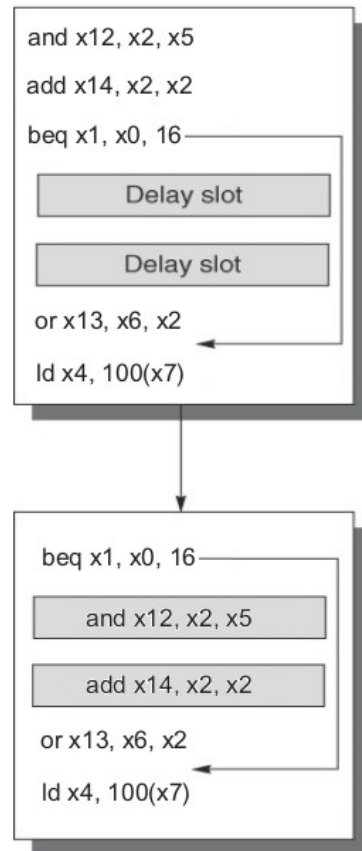


- a. Inserir bolhas (stalls) no pipeline até conhecer o endereço da próxima instrução a ser executada (flush instructions)



# Pipeline: dependências de controle

- b. Utilizar desvio atrasado (**delayed branch**)
- Técnica de software (compilador)
  - Utiliza o(s) ciclo(s) de clock seguinte(s) a instrução de desvio para colocar instruções do programa
  - Instruções inseridas não podem ter dependências com o desvio
  - Considerando 2 ciclos de atraso (2 delay slots) →

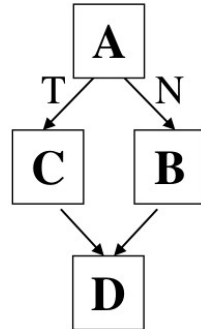


# Pipeline: dependências de controle

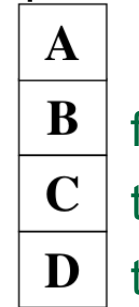


- c. Utilizar execução condicional (**predicated execution**)
- Idéia: Compilador converte dependência de controle em dependência de dado
  - Suporte no ISA pois instruções possuem um bit (predicado) indicando se a mesma será 'entregue' (committed); Exemplo ISA ARM
  - Somente instruções com predicado verdadeiro são entregues
  - Instrução 'descartada' vira um NOP

desvio normal



desvio predicado



f

t

t



# Pipeline: dependências de controle



## c. Utilizar execução condicional (predicated execution)

### Exemplo ISA ARM

31	28	27	16	15	8	7	0	
Cond	0	0	I	Opcode	S	Rn	Rd	Operand2
Cond	0	0	0	0	0	0	A S	Rd Rn Rs 1 0 0 1 Rm
Cond	0	0	0	0	1	U	A S	RdHi RdLo Rs 1 0 0 1 Rm
Cond	0	0	0	1	0	B	0 0	Rn Rd 0 0 0 0 1 0 0 1 Rm
Cond	0	1	I	P	U	B	W L	Rn Rd Offset
Cond	1	0	0	P	U	S	W L	Rn Register List
Cond	0	0	0	P	U	1	W L	Rn Rd Offset1 1 S H 1 Offset2
Cond	0	0	0	P	U	0	W L	Rn Rd 0 0 0 0 1 S H 1 Rm

#### Instruction type

Data processing / PSR Transfer

Multiply

Long Multiply (v3M / v4 only)

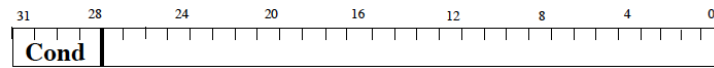
Swap

Load/Store Byte/Word

Load/Store Mult:

Halfword transfer: Imm

Halfword transfer: Reg



0000 = EQ - Z set (equal)

0001 = NE - Z clear (not equal)

0010 = HS / CS - C set (unsigned higher or same)

0011 = LO / CC - C clear (unsigned lower)

0100 = MI - N set (negative)

0101 = PL - N clear (positive or zero)

0110 = VS - V set (overflow)

0111 = VC - V clear (no overflow)

1000 = HI - C set and Z clear

1001 = LS - C clear or Z (set unsigned lower or same)

1010 = GE - N set and V set, or N clear and V clear (> or =)

1011 = LT - N set and V clear, or N clear and V set (>)

1100 = GT - Z clear, and either N set and V set, or N clear and V set (>)

1101 = LE - Z set, or N set and V clear, or N clear and V set (<, or =)

1110 = AL - always

1111 = NV - reserved.

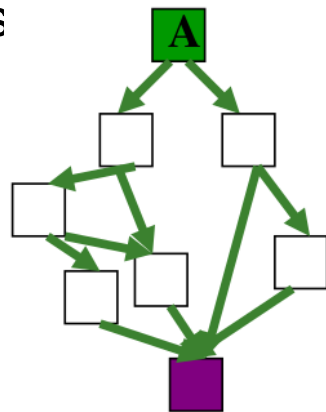
# Pipeline: dependências de controle



- c. Utilizar execução condicional (**predicated execution**)

Problemas:

- a) **Adaptividade**: não adaptativa ao comportamento dos desvios (switch-case; if-elseif-else)
- b) **CFG complexos**: não aplicável para laços ou grafos com fluxos complexos (CFG)
- c) **ISA**: Requer mudanças profundas no ISA



# Pipeline: dependências de controle



É possível fazer melhor que colocar bolhas no Pipeline?

~20% das instruções nos programas são instruções de controle de fluxo

~50% dos desvios 'para frente' são tomados (i.e., if-then-else)

~90% dos desvios 'para tras' são tomados (i.e., loop back)

No geral os desvios, tipicamente:

~70% são tomados

~30% não tomados

[Lee and Smith, 1984]

# Pipeline: dependências de controle



d. Utilizar predição de desvio

- Predição estática (compilação)

Estratégias:

- Sempre tomado (always taken)
- Nunca tomado (never taken)
- tomado para trás; não tomado para frente (BTFN)
- baseado em profile

# Pipeline: dependências de controle



d. Utilizar predição de desvio

- Predição estática

Ao invés de esperar a dependência verdadeira ser resolvida prever

$\text{'nextPC = PC+4'}$

mantendo a busca de instruções funcionando e o pipeline cheio

É uma boa previsão?

Expectativa:  $\text{'nextPC = PC+4'}$  ~86% do tempo. E os outros ~14%?

# Pipeline: dependências de controle



d. Utilizar predição de desvio

- **Predição dinâmica**

- Vantagens:

- Predição baseada na história de execução dos desvios
    - Pode se adaptar dinamicamente as mudanças de comportamento da aplicação
    - não precisa conhecer o perfil das aplicações para garantir boas taxas de acerto

- Desvantagem:

- Mais complexo (requer hardware adicional)

# Pipeline: dependências de controle



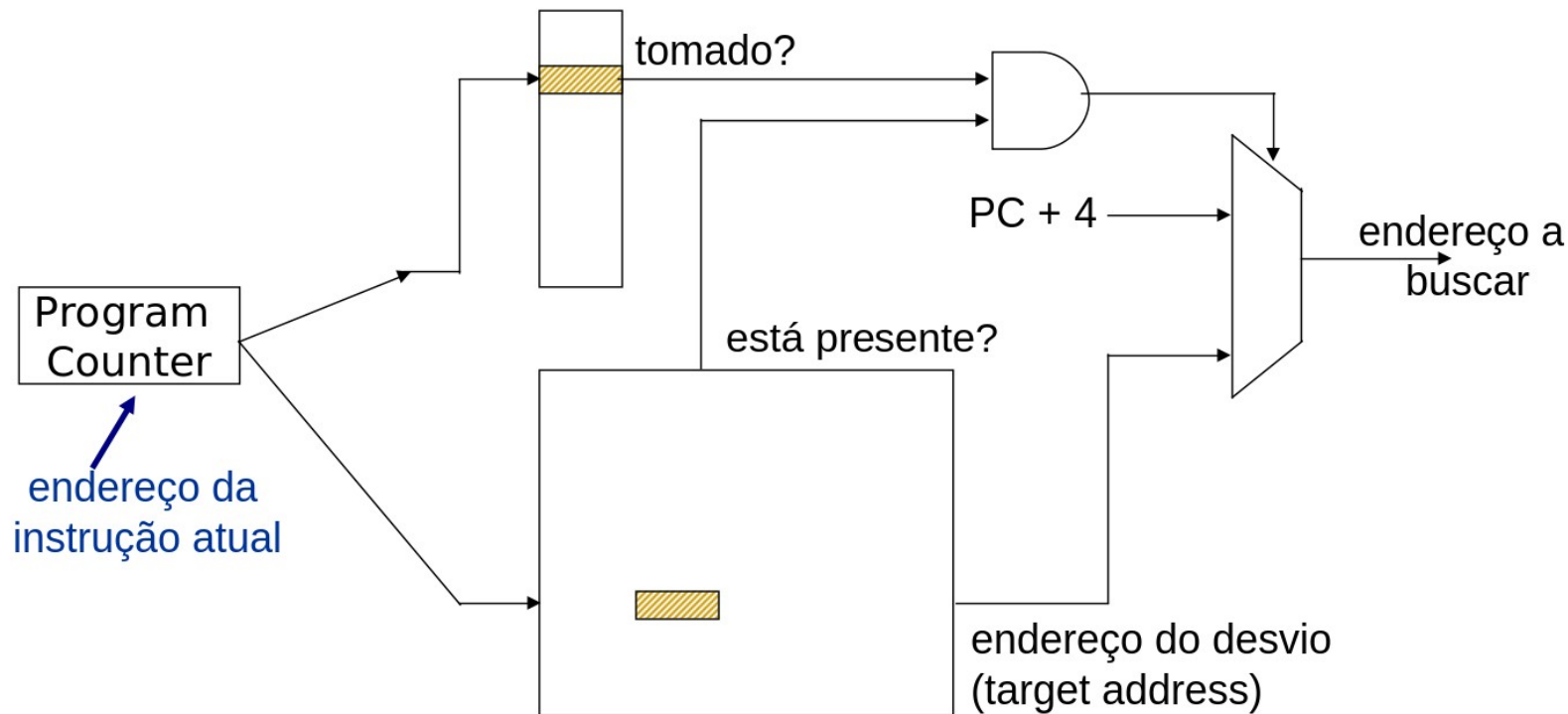
d. Utilizar predição de desvio

- Possui três requisitos para ser alcançado na etapa de busca
  - saber que é uma instrução de desvio condicional
  - a direção do desvio (predição)
  - o endereço do desvio (target address), se for tomado
- O endereço do desvio permanece o mesmo para desvios condicionais em todas as suas execuções
- Idéia: armazenar o endereço de desvio calculado em uma execução anterior e vincular o mesmo ao PC

# Pipeline: dependências de controle



d. Utilizar predição de desvio





# Pipeline: dependências de controle



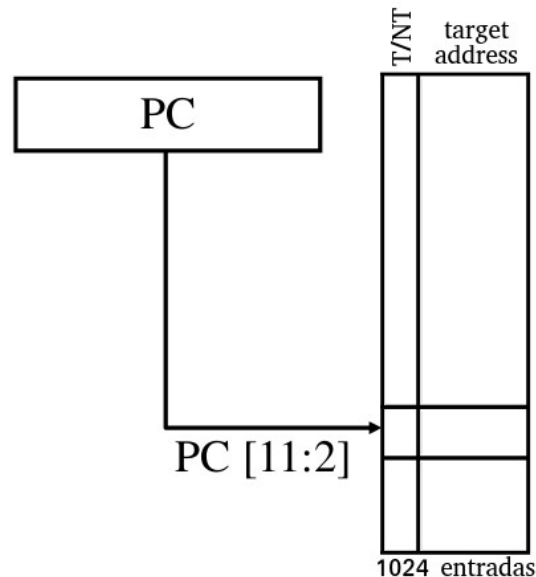
d. Utilizar predição de desvio

- Predição dinâmica

→ Last time predictor

- Branch History Table (ou Branch Target Table)

- entrada na memória definida pela instrução de desvio (branch)
- memória indexada pelos bits menos significativos de PC (sem rótulo); contém alias
- 1 bit de predição armazena o resultado do último desvio (1 - T / 0 - NT)



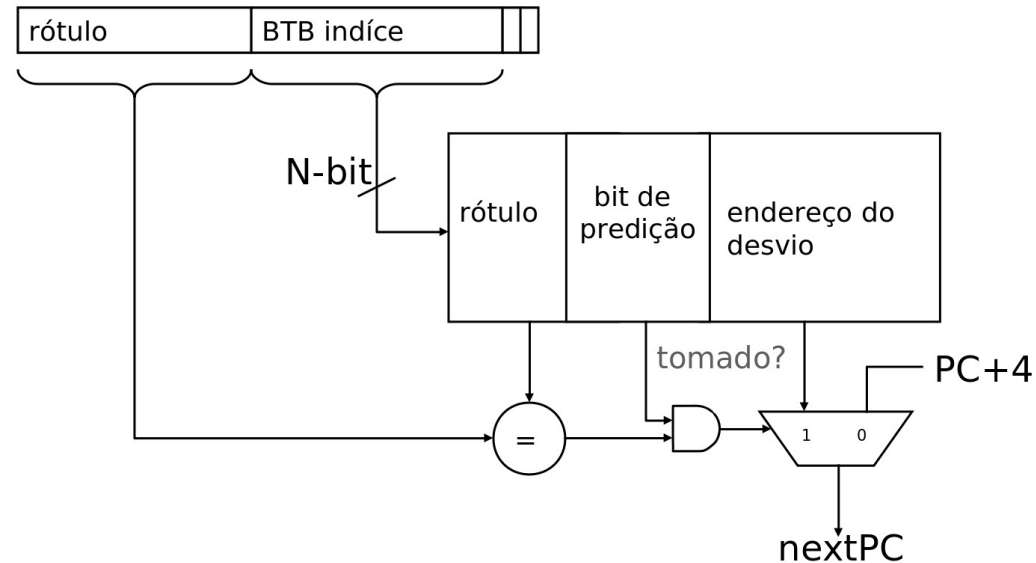
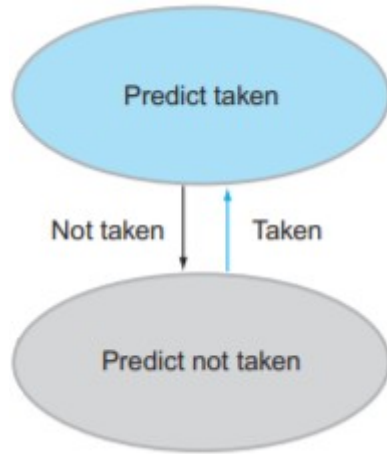
# Pipeline: dependências de controle



d. Utilizar previsão de desvio

- Predição dinâmica

→ Last time prediction



# Pipeline: dependências de controle



d. Utilizar predição de desvio

- **Predição dinâmica**

→ Last time prediction

Problema: O last time predictor muda sua predição a cada erro. Isto leva a muitas situações de dois erros consecutivos. Ex: laços

Solução: Adicionar uma histerese no preditor evitando que ele mude a cada erro

Usar dois bits de predição ao invés de apenas 1

Deve errar duas vezes consecutivas para trocar a previsão

# Pipeline: dependências de controle



d. Utilizar previsão de desvio

- **Predição dinâmica**

→ Two bit counter predictor (preditor bimodal)

- ~90-95% de acerto para muitas aplicações

