



Universidade Federal da Fronteira Sul
Curso de Ciência da Computação
Campus Chapecó



GEX 612 – Organização de Computadores

RISC-V

Implementação Pipeline

Prof. Luciano L. Caimi
lcaimi@uffs.edu.br

Recursos

- Simulador Ripes



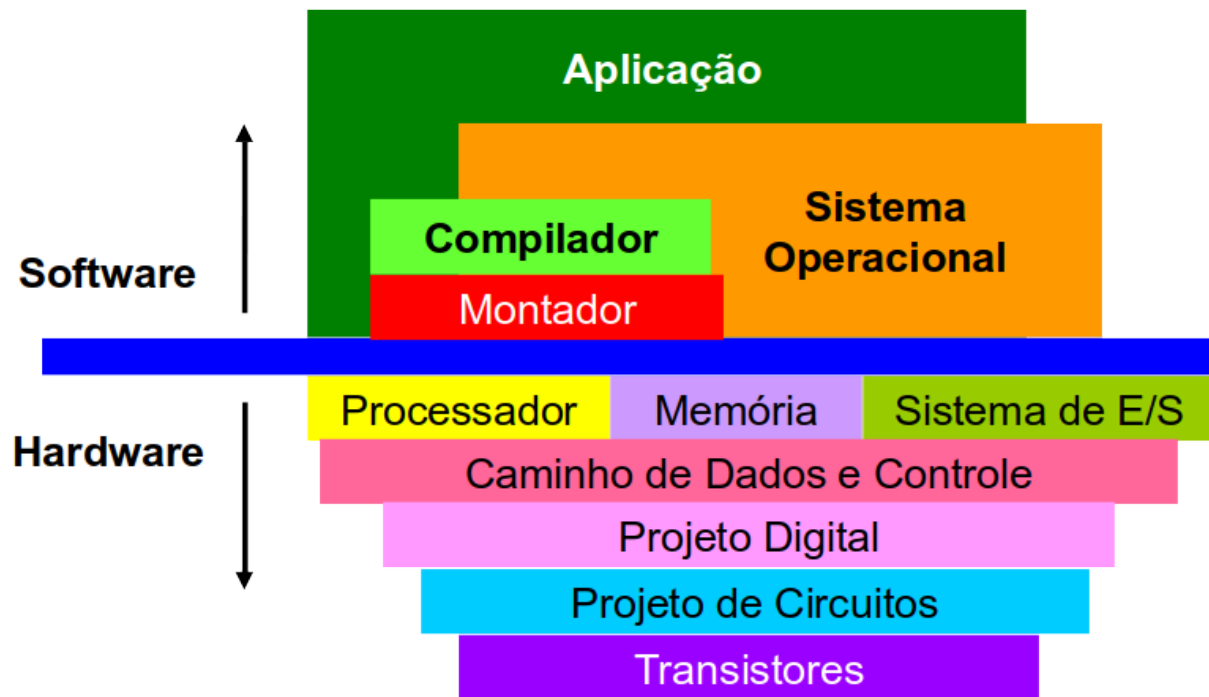
<https://github.com/mortbopet/Ripes>

- Simulador WebRISC-V

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



Introdução: Arquitetura Multinível



Introdução: RISC-V - ISA



Formato das Instruções:

32-bit RISC-V Instruction Formats

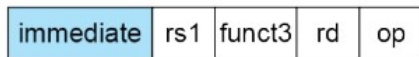
Instruction Formats	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Register/register	funct7							rs2				rs1				funct3		rd				opcode										
Immediate	imm[11:0]												rs1				funct3		rd				opcode									
Upper Immediate	imm[31:12]																				rd				opcode							
Store	imm[11:5]							rs2				rs1				funct3		imm[4:0]				opcode										
Branch	[12]	imm[10:5]						rs2				rs1				funct3		imm[4:1]			[11]	opcode										
Jump	[20]	imm[10:1]										[11]	imm[19:12]							rd				opcode								

- **opcode (7 bit):** partially specifies which of the 6 types of *instruction formats*
- **funct7 + funct3 (10 bit):** combined with **opcode**, these two fields describe what operation to perform
- **rs1 (5 bit):** specifies register containing first operand
- **rs2 (5 bit):** specifies second register operand
- **rd (5 bit):** Destination register specifies register which will receive result of computation

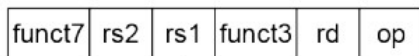
Introdução: RISC-V - modos de endereçamento



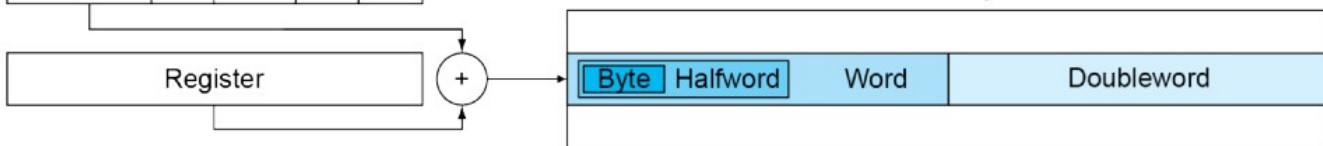
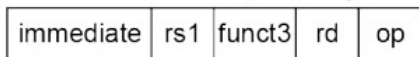
1. Immediate addressing



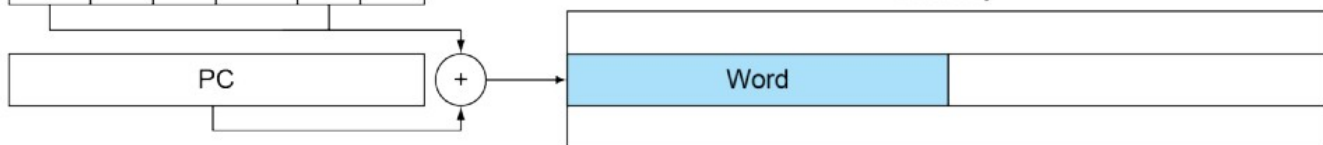
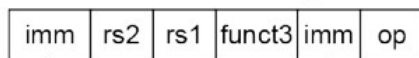
2. Register addressing



3. Base addressing, i.e., displacement addressing



4. PC-relative addressing



Introdução: RISC-V - ISA

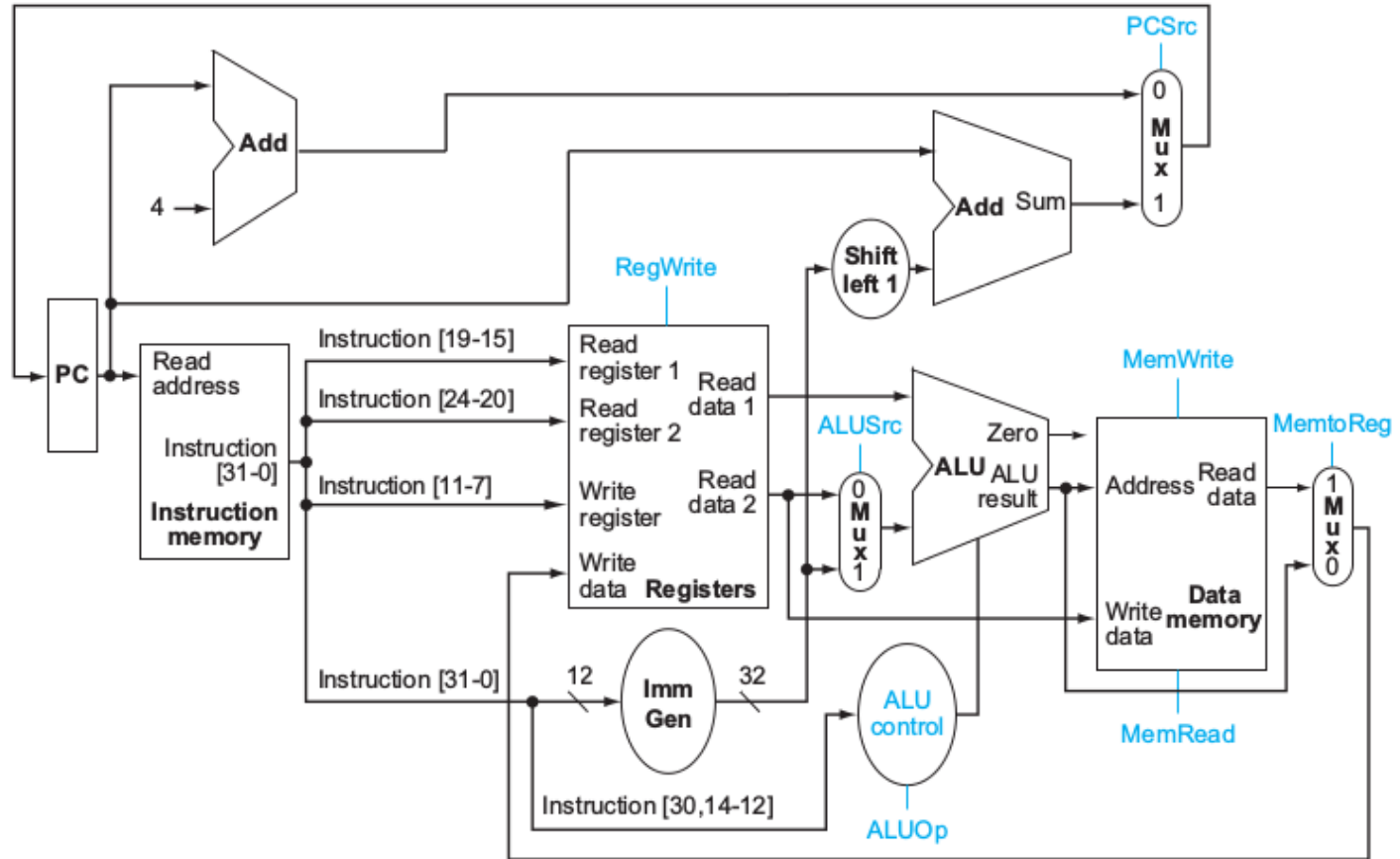


Instruções RV32I

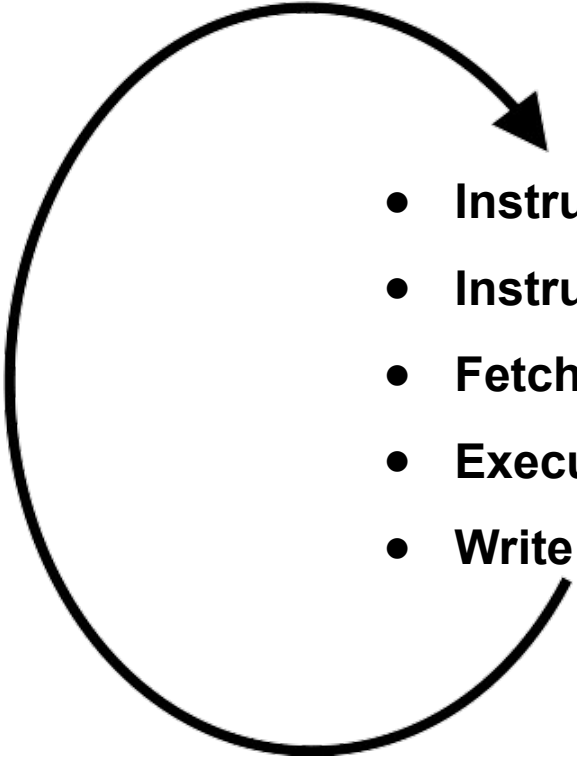
imm[31:12]					rd	0110111	LUI
imm[31:12]					rd	0010111	AUIPC
imm[20:10:1 11 19:12]					rd	1101111	JAL
imm[11:0]			rs1	000	rd	1100111	JALR
imm[12:10:5]		rs2	rs1	000	imm[4:1:11]	1100011	BEQ
imm[12:10:5]		rs2	rs1	001	imm[4:1:11]	1100011	BNE
imm[12:10:5]		rs2	rs1	100	imm[4:1:11]	1100011	BLT
imm[12:10:5]		rs2	rs1	101	imm[4:1:11]	1100011	BGE
imm[12:10:5]		rs2	rs1	110	imm[4:1:11]	1100011	BLTU
imm[12:10:5]		rs2	rs1	111	imm[4:1:11]	1100011	BGEU
imm[11:0]			rs1	000	rd	0000011	LB
imm[11:0]			rs1	001	rd	0000011	LH
imm[11:0]			rs1	010	rd	0000011	LW
imm[11:0]			rs1	100	rd	0000011	LBU
imm[11:0]			rs1	101	rd	0000011	LHU
imm[11:5]		rs2	rs1	000	imm[4:0]	0100011	SB
imm[11:5]		rs2	rs1	001	imm[4:0]	0100011	SH
imm[11:5]		rs2	rs1	010	imm[4:0]	0100011	SW

imm[11:0]			rs1	000	rd	0010011	ADDI
imm[11:0]			rs1	010	rd	0010011	SLTI
imm[11:0]			rs1	011	rd	0010011	SLTIU
imm[11:0]			rs1	100	rd	0010011	XORI
imm[11:0]			rs1	110	rd	0010011	ORI
imm[11:0]			rs1	111	rd	0010011	ANDI
0000000		shamt	rs1	001	rd	0010011	SLLI
0000000		shamt	rs1	101	rd	0010011	SRLI
0100000		shamt	rs1	101	rd	0010011	SRAI
0000000		rs2	rs1	000	rd	0110011	ADD
0100000		rs2	rs1	000	rd	0110011	SUB
0000000		rs2	rs1	001	rd	0110011	SLL
0000000		rs2	rs1	010	rd	0110011	SLT
0000000		rs2	rs1	011	rd	0110011	SLTU
0000000		rs2	rs1	100	rd	0110011	XOR
0000000		rs2	rs1	101	rd	0110011	SRL
0100000		rs2	rs1	101	rd	0110011	SRA
0000000		rs2	rs1	110	rd	0110011	OR
0000000		rs2	rs1	111	rd	0110011	AND
0000	pred	succ	00000	000	00000	0001111	FENCE
0000	0000	0000	00000	001	00000	0001111	FENCE.I
000000000000			00000	000	00000	1110011	ECALL
000000000001			00000	000	00000	1110011	EBREAK
csr			rs1	001	rd	1110011	CSRWR
csr			rs1	010	rd	1110011	CSRWS
csr			rs1	011	rd	1110011	CSRWC
csr			zimm	101	rd	1110011	CSRWI
csr			zimm	110	rd	1110011	CSRWSI
csr			zimm	111	rd	1110011	CSRWC

Bloco operativo monociclo

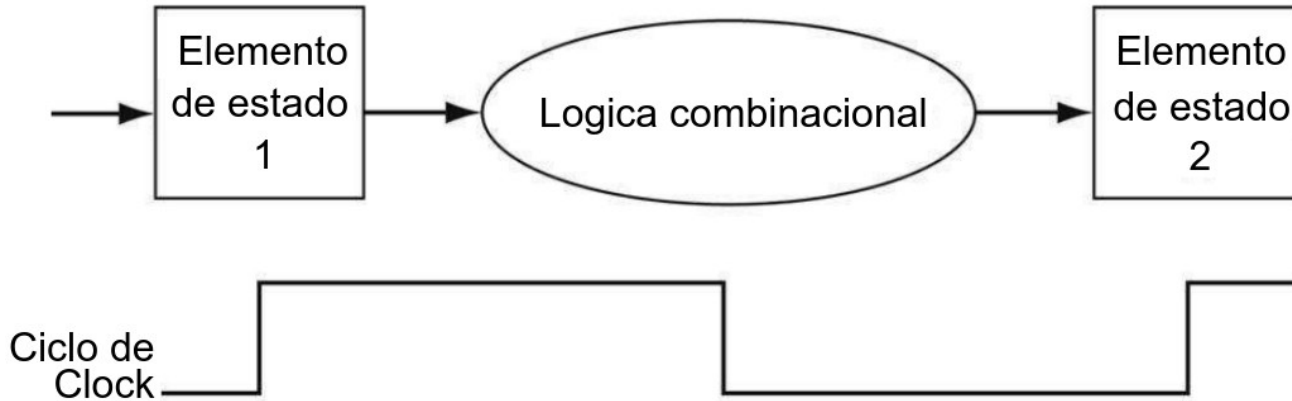


Ciclo de Instrução

- 
- A large, thick black curved arrow starts at the bottom of the list and points upwards to the first item, 'Instruction Fetch - IF', indicating a continuous loop.
- **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)**

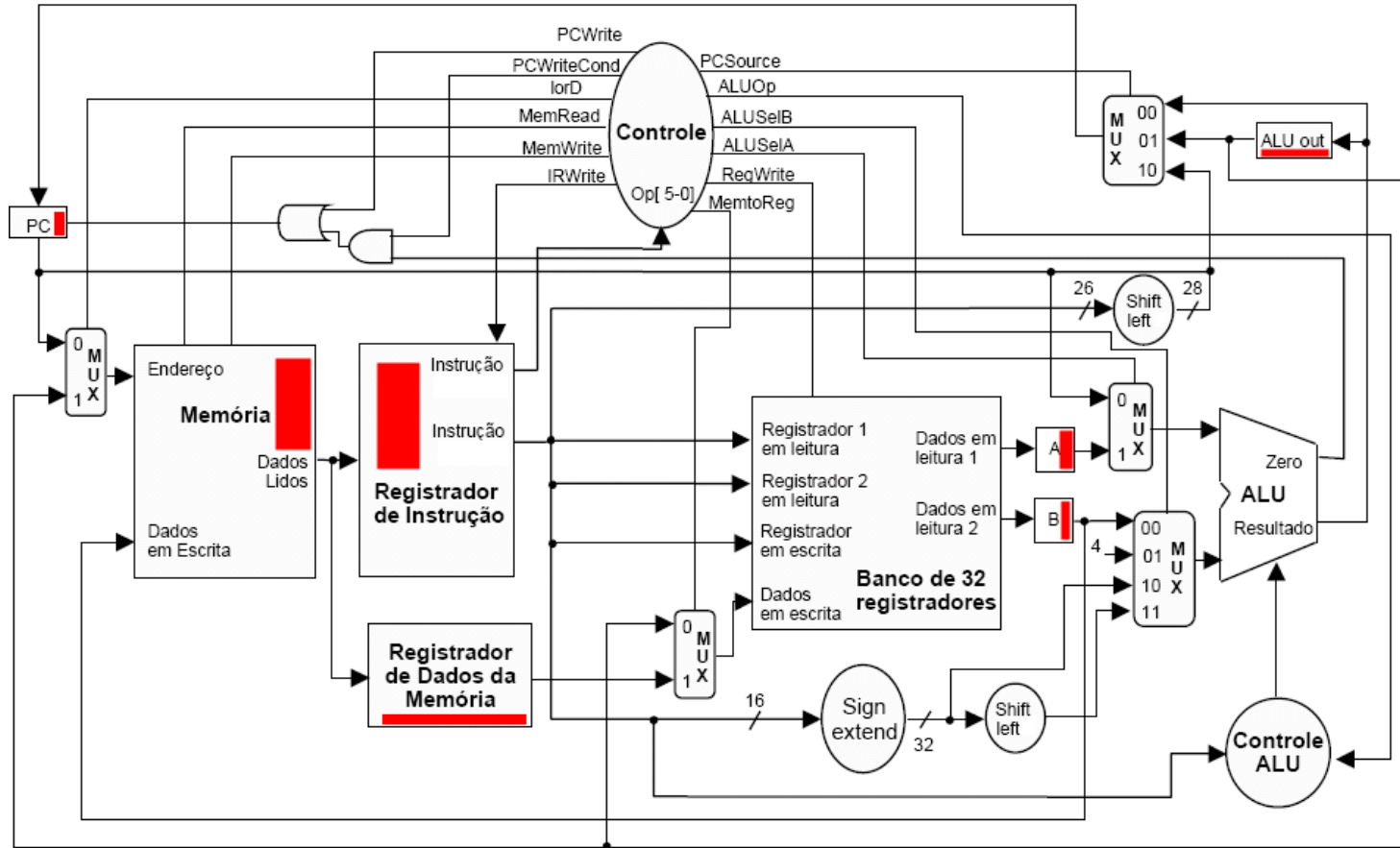
Ciclo de processamento

- Elementos de armazenamentos gatilhados na borda de subida do clock

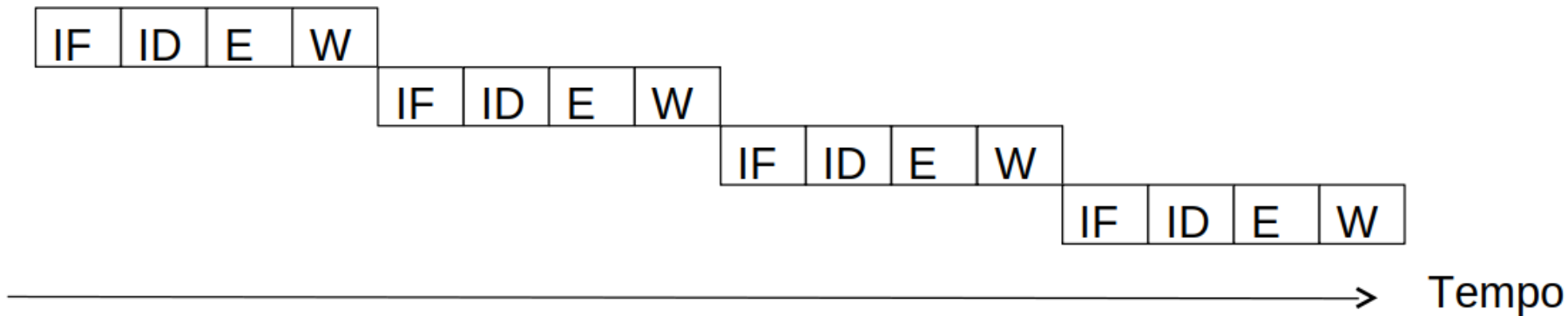


- Valor é armazenado no final do ciclo de clock anterior e lido no início do clock seguinte
- Saída é igual ao valor armazenado no elemento (não é necessário permissão para ler o valor)

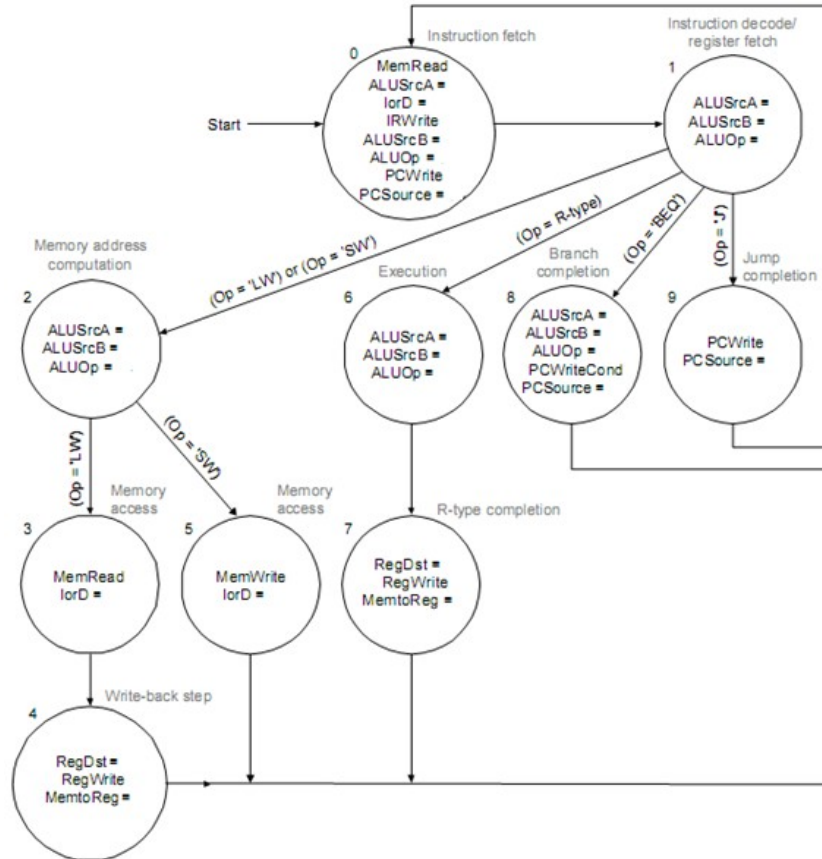
Bloco operativo multiciclo



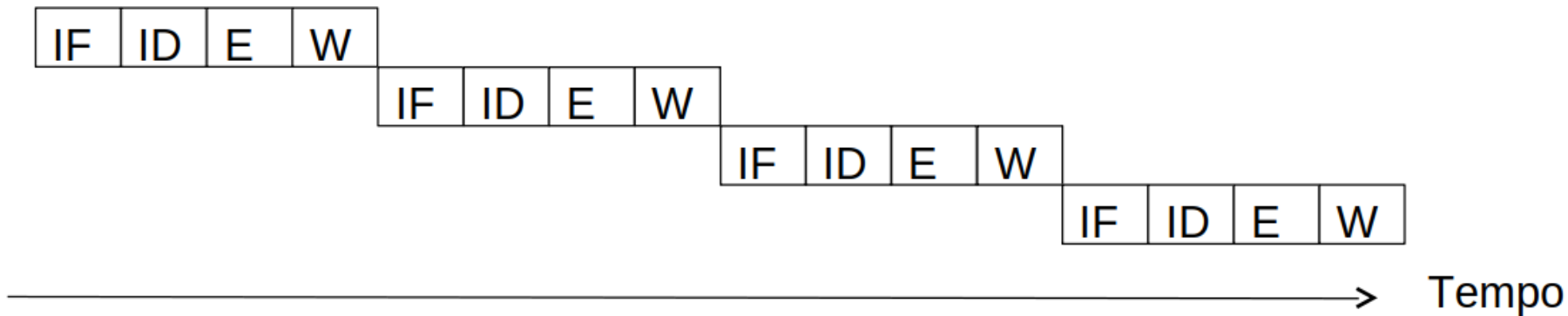
Implementação multiciclo



Controle multiciclo

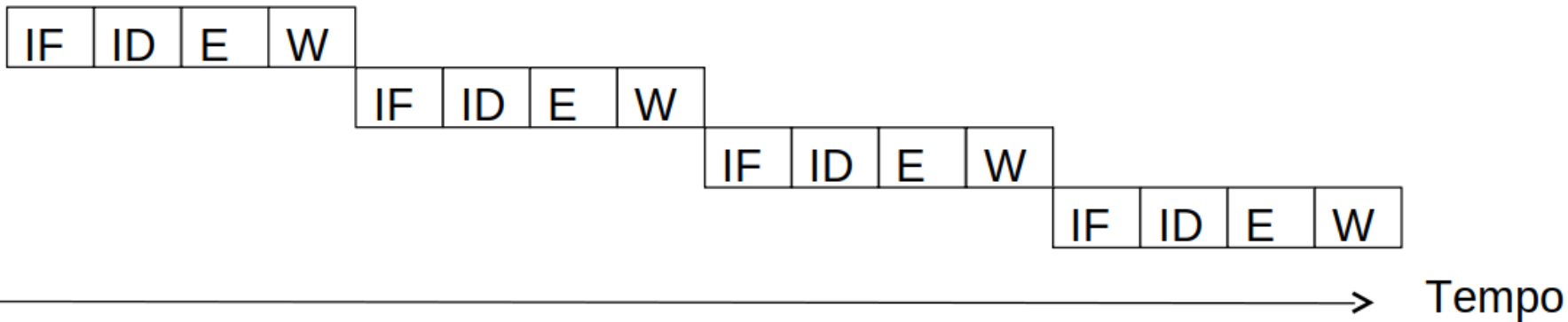


Implementação multiciclo



- Quais as limitações da operação multiciclo?

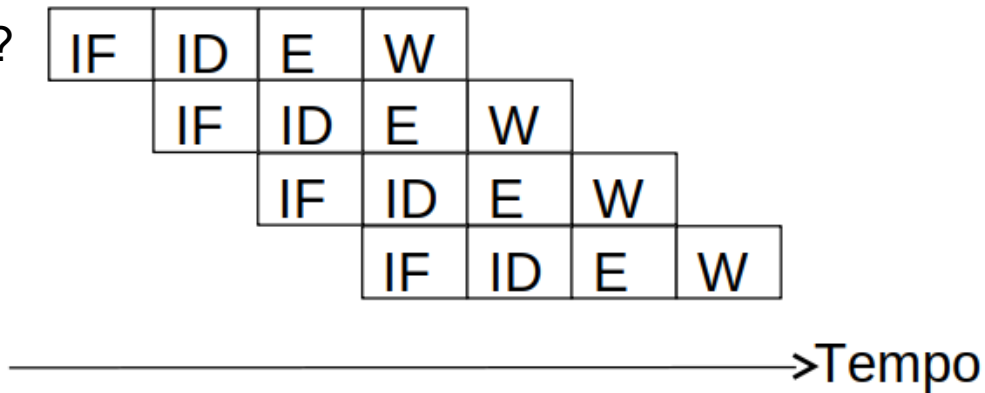
Pipeline



- É possível otimizar a implementação?

SIM

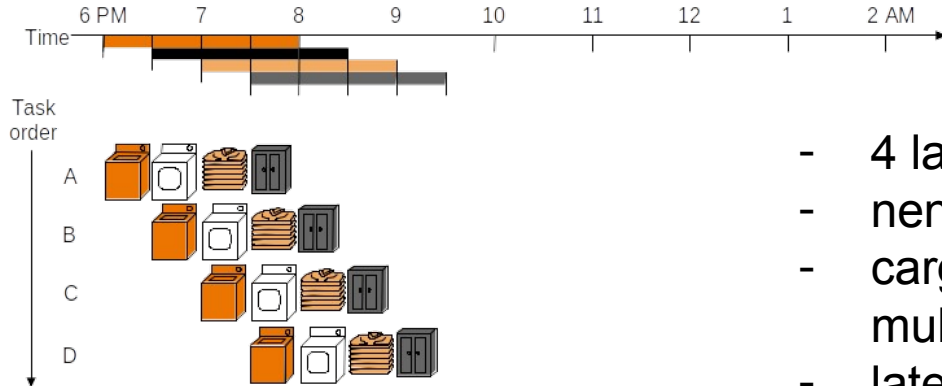
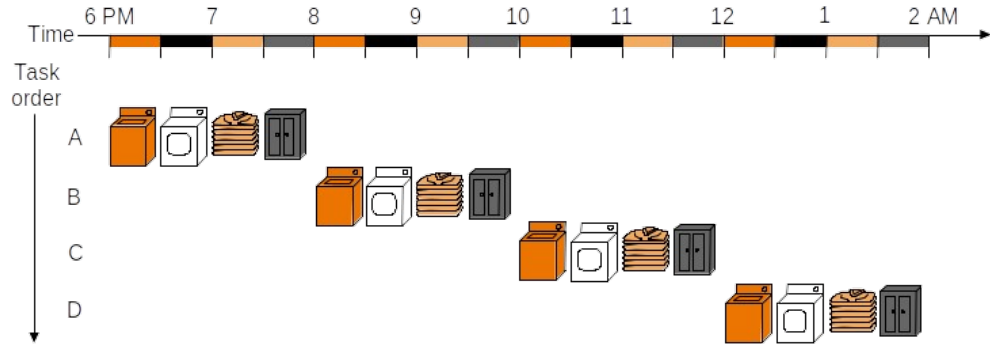
Paralelismo no nível de Instrução
(Instruction Level Parallelism - ILP)



Pipeline

Analogia da Lavanderia

- caso ótimo

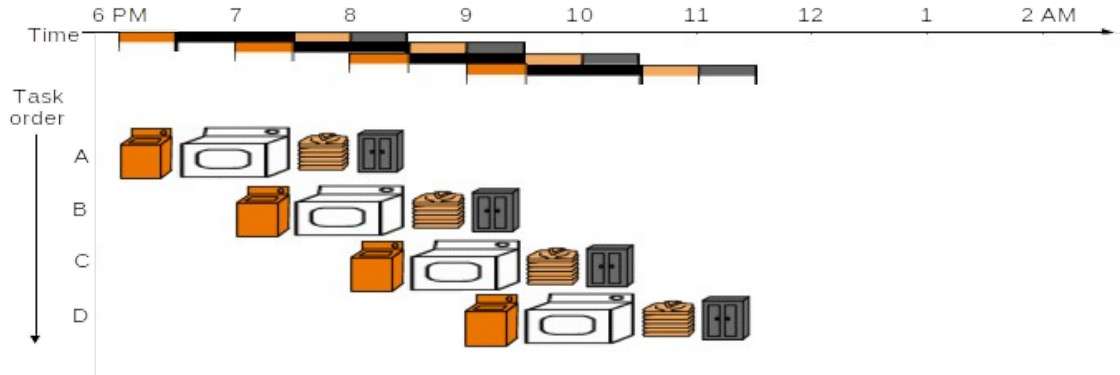
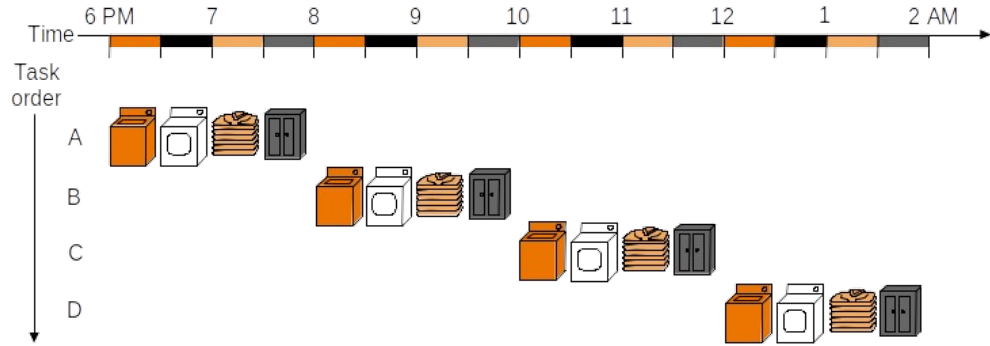


- 4 lavagens em paralelo
- nenhum novo recurso adicionado
- carga de trabalho (throughput) realizado multiplicado por 4
- latencia por lavagem é a mesma

Pipeline

Analogia da Lavanderia

- caso real (1)

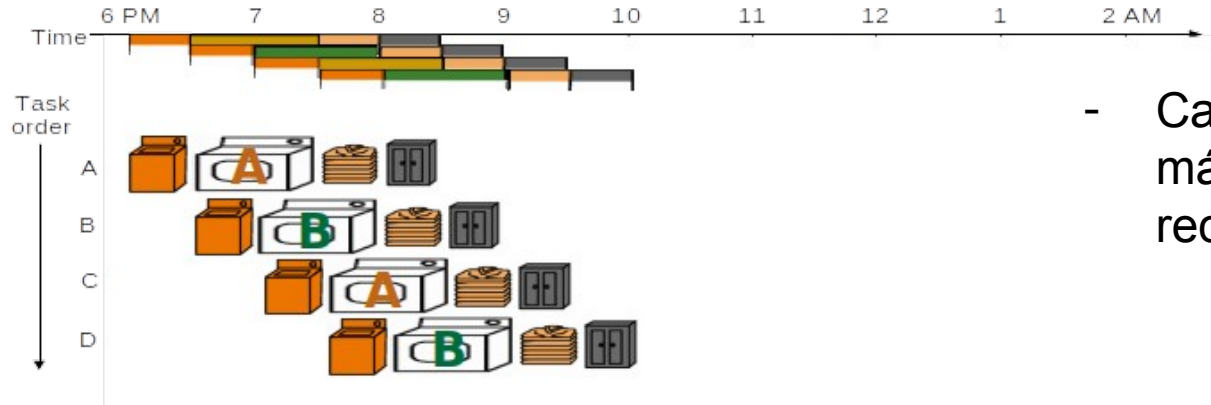
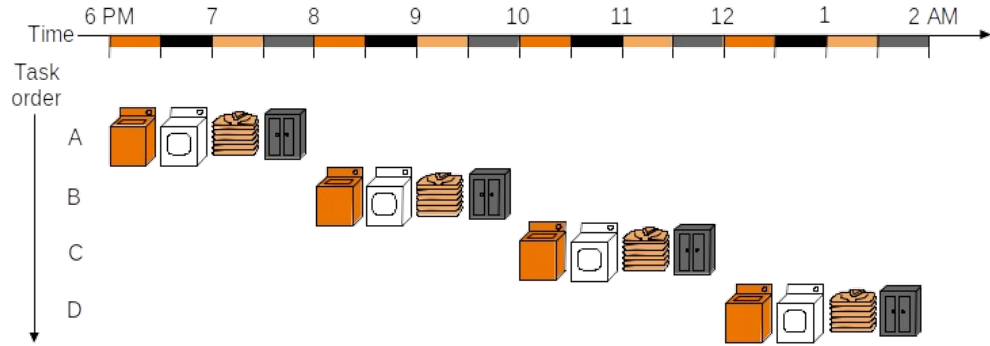


- o estágio mais lento (máquina de lavar) define a evolução do pipeline e consequentemente a carga de trabalho (throughput)

Pipeline

Analogia da Lavanderia

- caso real (2)

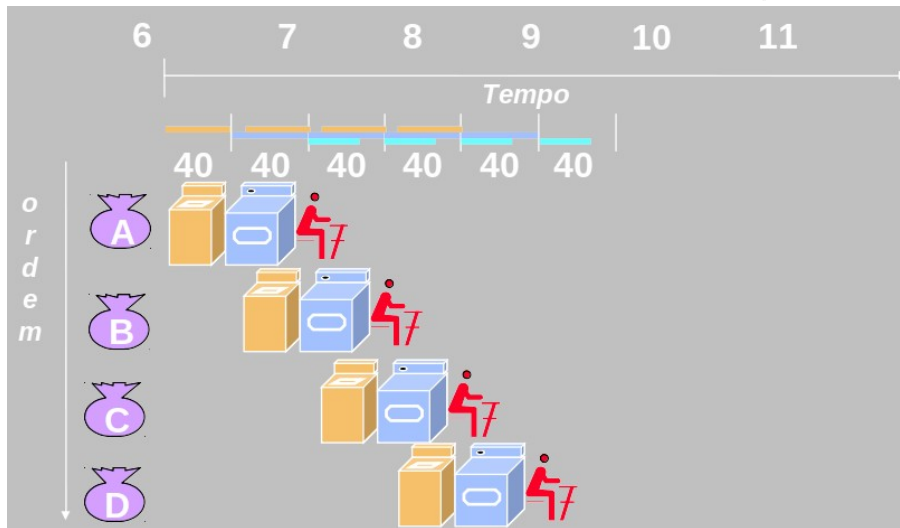
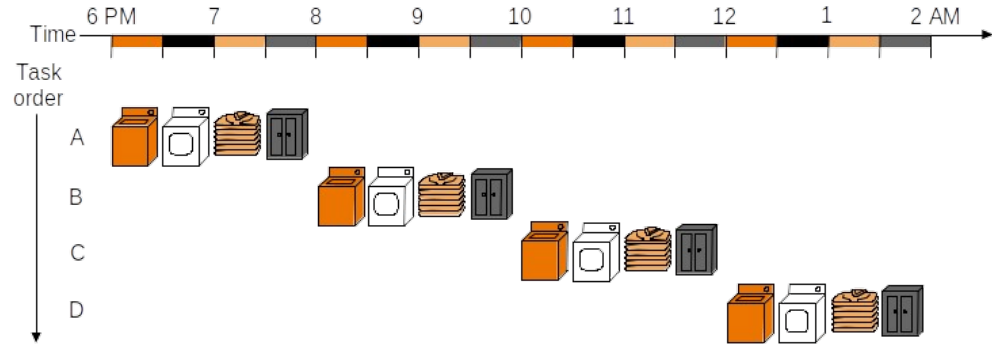


- Carga de trabalho (throughput) máxima com a duplicação de recursos (máquina de lavar)

Pipeline

Analogia da Lavanderia

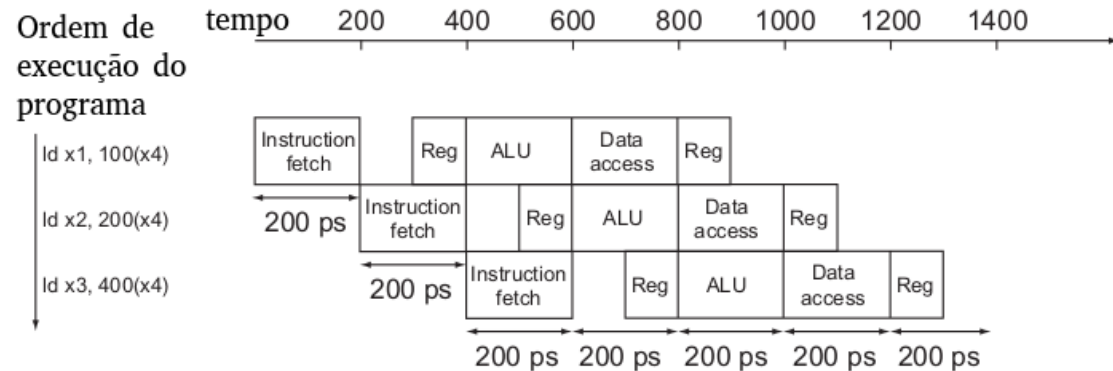
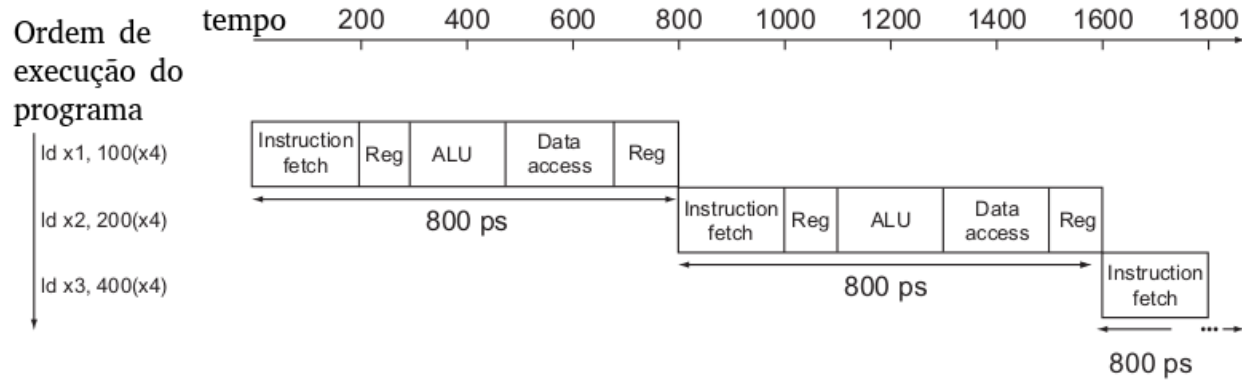
- caso real (3)



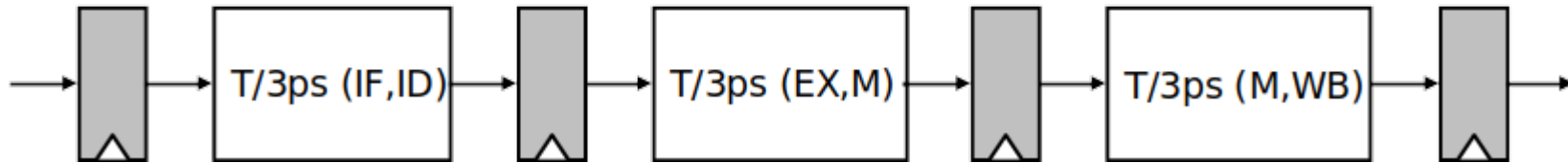
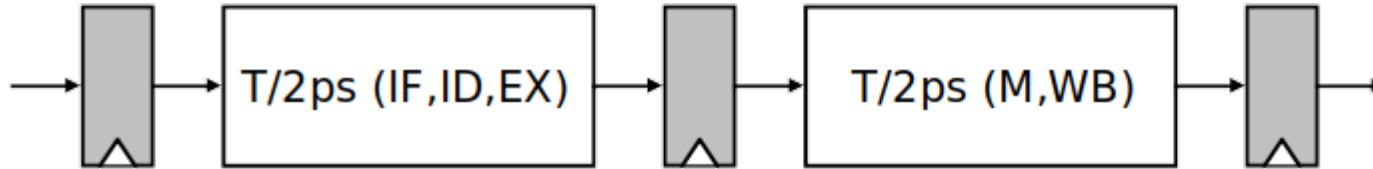
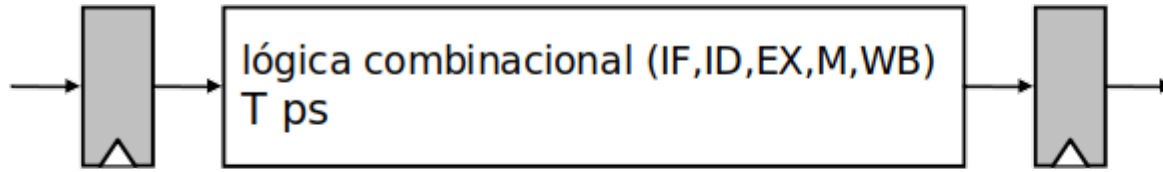
- Balanceamento entre estágios nem sempre é possível e estágio mais lento define a carga de trabalho (throughput)

- **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

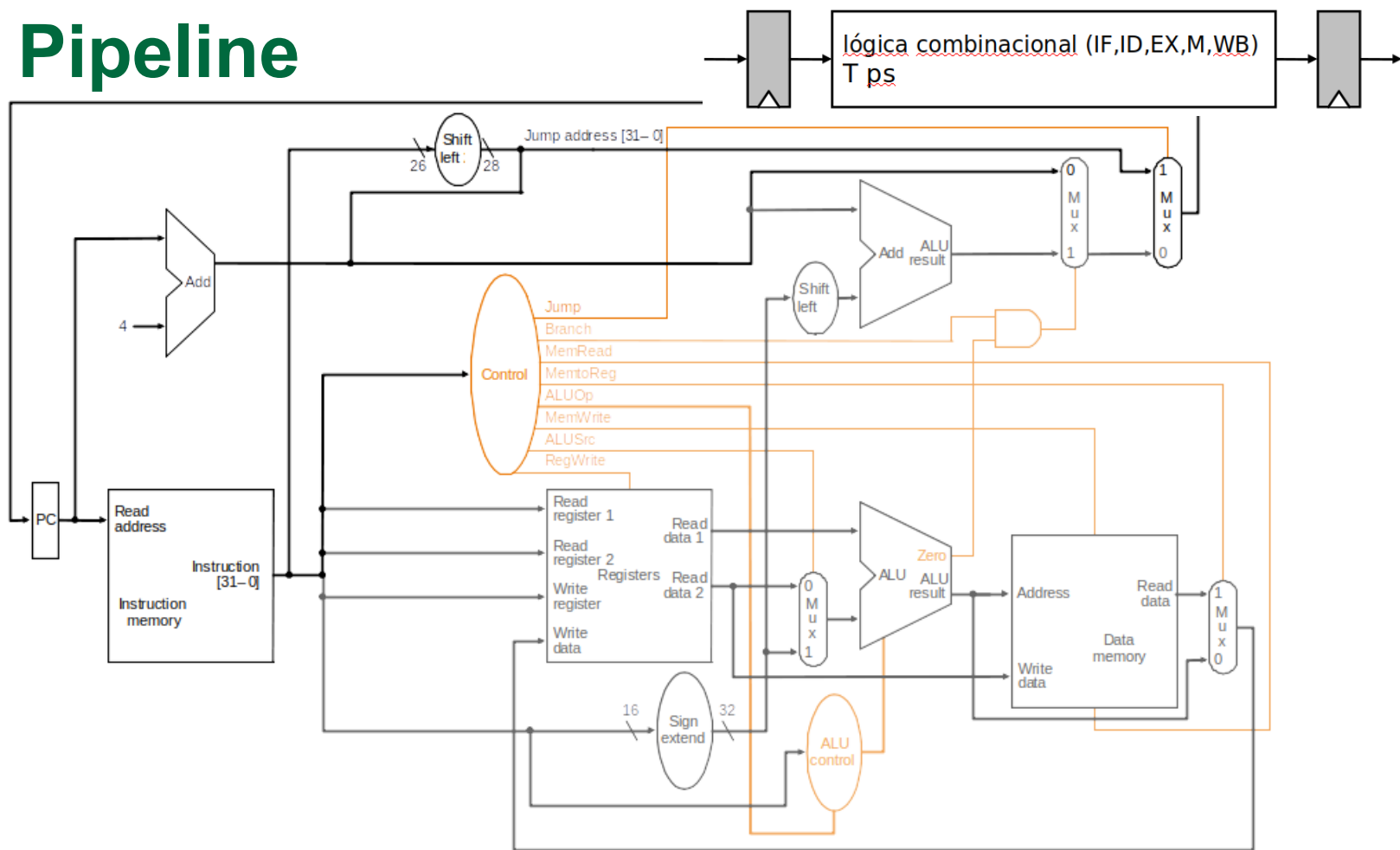
Pipeline



Pipeline

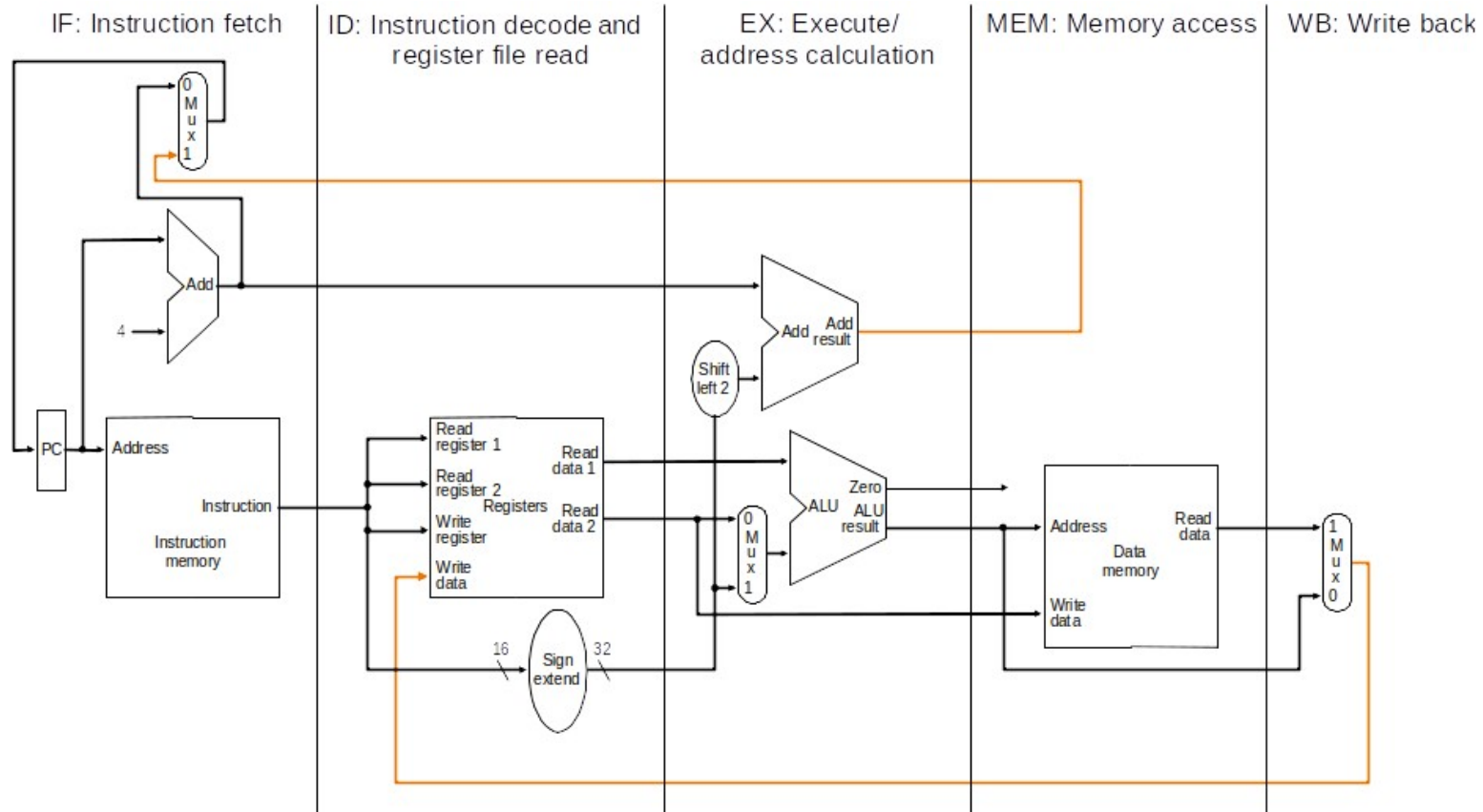


Pipeline



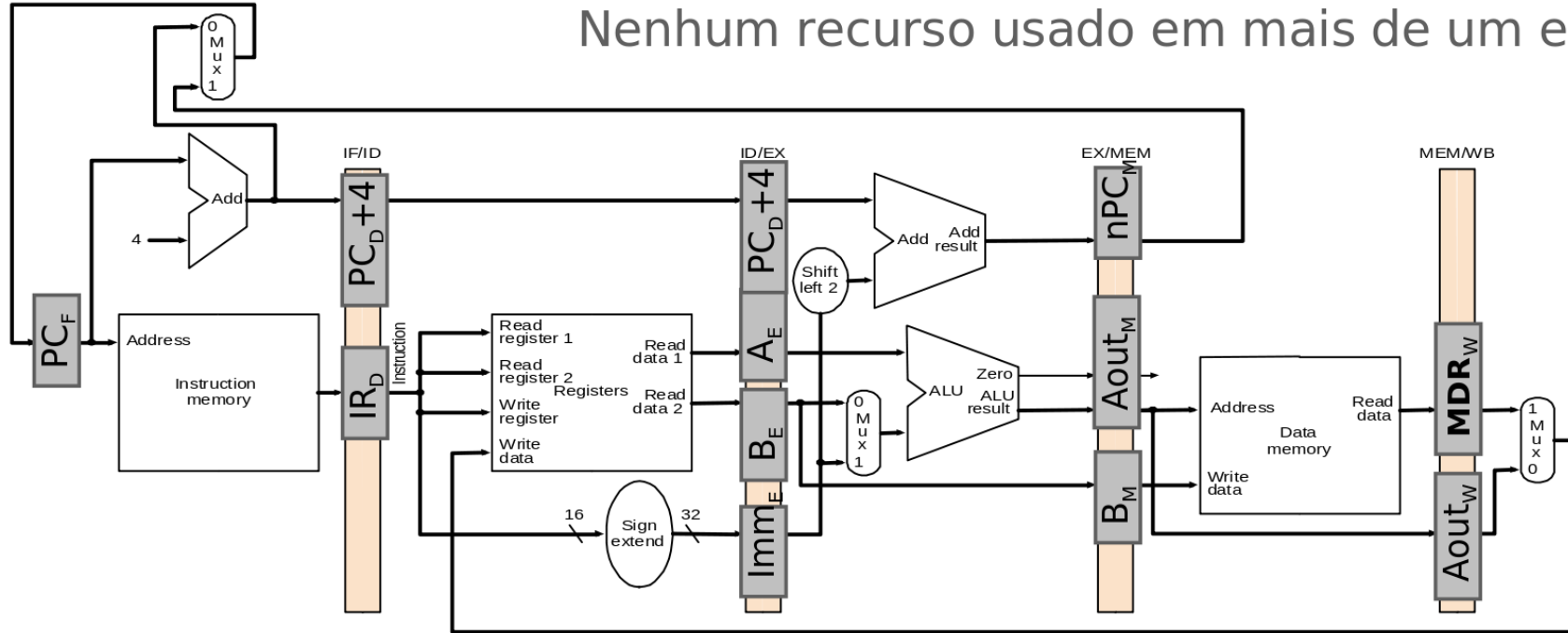
Pipeline 5 estágios

O que acontece em cada estágio???



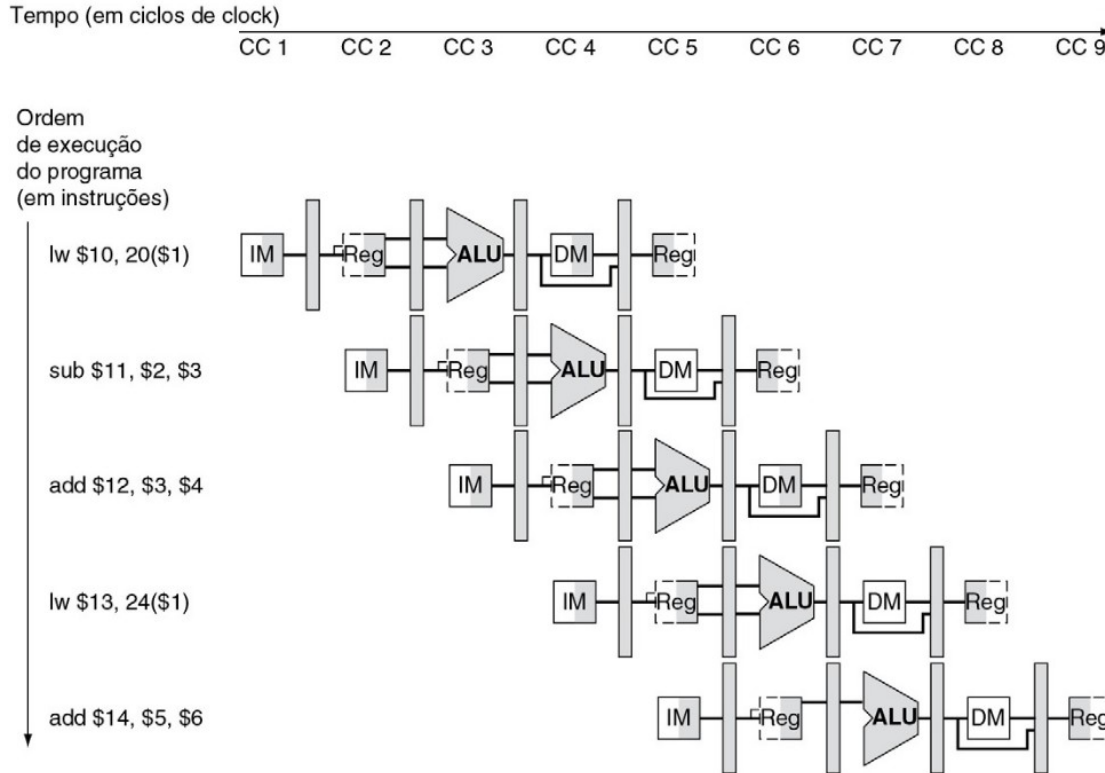
Pipeline 5 estágios

Nenhum recurso usado em mais de um estágio



Pipeline

Representação do Pipeline:



Pipeline

Estágios: IF, ID, EX, MA, WB



Exemplo:

$$S = \underline{(A + 2) * D}$$
$$(C - 5)$$

```
.text
lw    t1, C
lw    t0, A
lw    t2, D
addi  s1, t1, -5
la    t3, S
addi  s0, t0, 2
mul   s2, s0, t2
div   s2, s2, s1
sw    s2, 0 (t3)
```

	CC1	CC2	CC3	CC4	CC5	CC6	CC7	CC8
lw C								
lw A								
lw D								
addi								
la**								
addi								
mul*								
div*								
sw								

** div e mul podem levar mais de um ciclo de clock dependendo da implementação

** la é uma pseudo-instrução convertida em duas instruções de máquina (ignoramos isto no exemplo)

Pipeline

Estágios: IF, ID, EX, MA, WB

Exemplo:

$$S = \frac{(A + 2) * D}{(C - 5)}$$

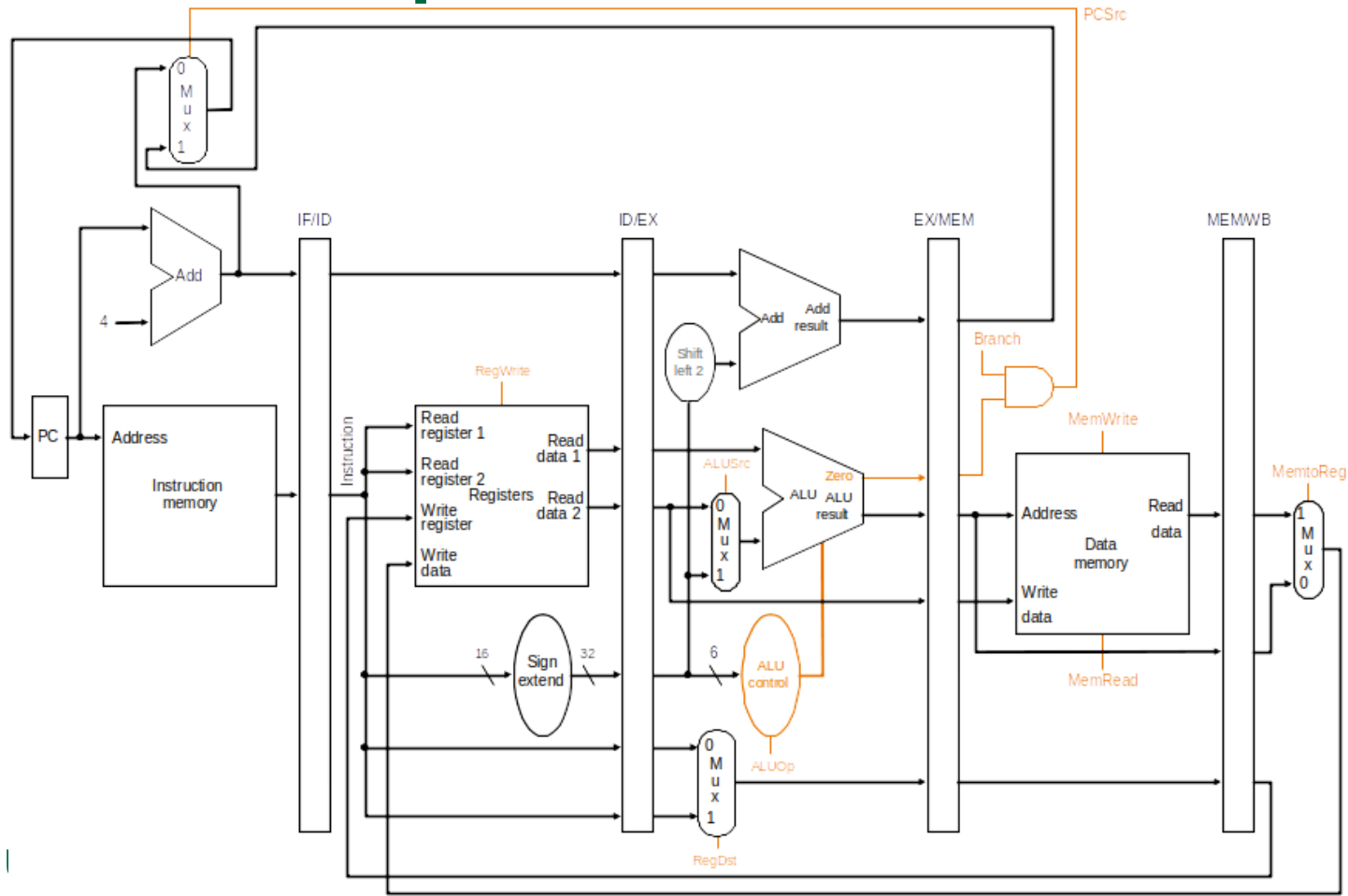
```
.text
lw    t1, C
lw    t0, A
lw    t2, D
addi  s1, t1, -5
la    t3, S
addi  s0, t0, 2
mul   s2, s0, t2
div   s2, s2, s1
sw    s2, 0 (t3)
```

	CC1	CC2	CC3	CC4	CC5	CC6	CC7	CC8	CC9
lw C	IF	ID	EX	MA	WB				
lw A		IF	ID	EX	MA	WB			
lw D			IF	ID	EX	MA	WB		
addi				IF	ID	EX	MA	WB	WB
la**					IF	ID	EX	MA	WB
addi						IF	ID	EX	MA
mul*							IF	ID	EX
div*								IF	ID
sw									

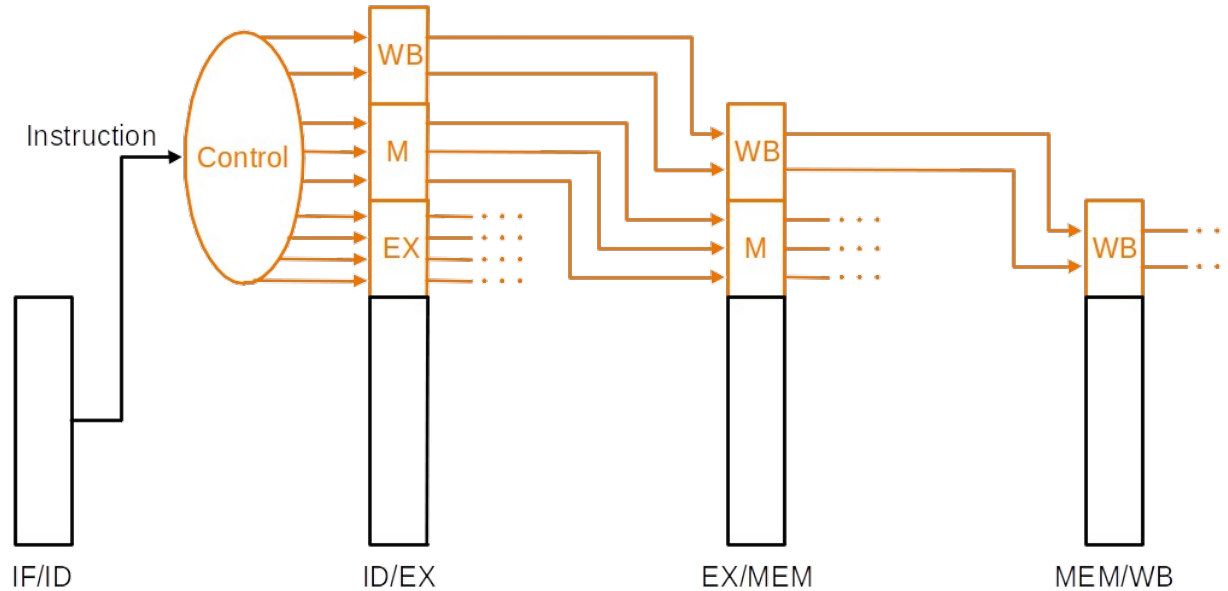
** div e mul podem levar mais de um ciclo de clock dependendo da implementação

** la é uma pseudo-instrução convertida em duas instruções de máquina (ignoramos isto no exemplo)

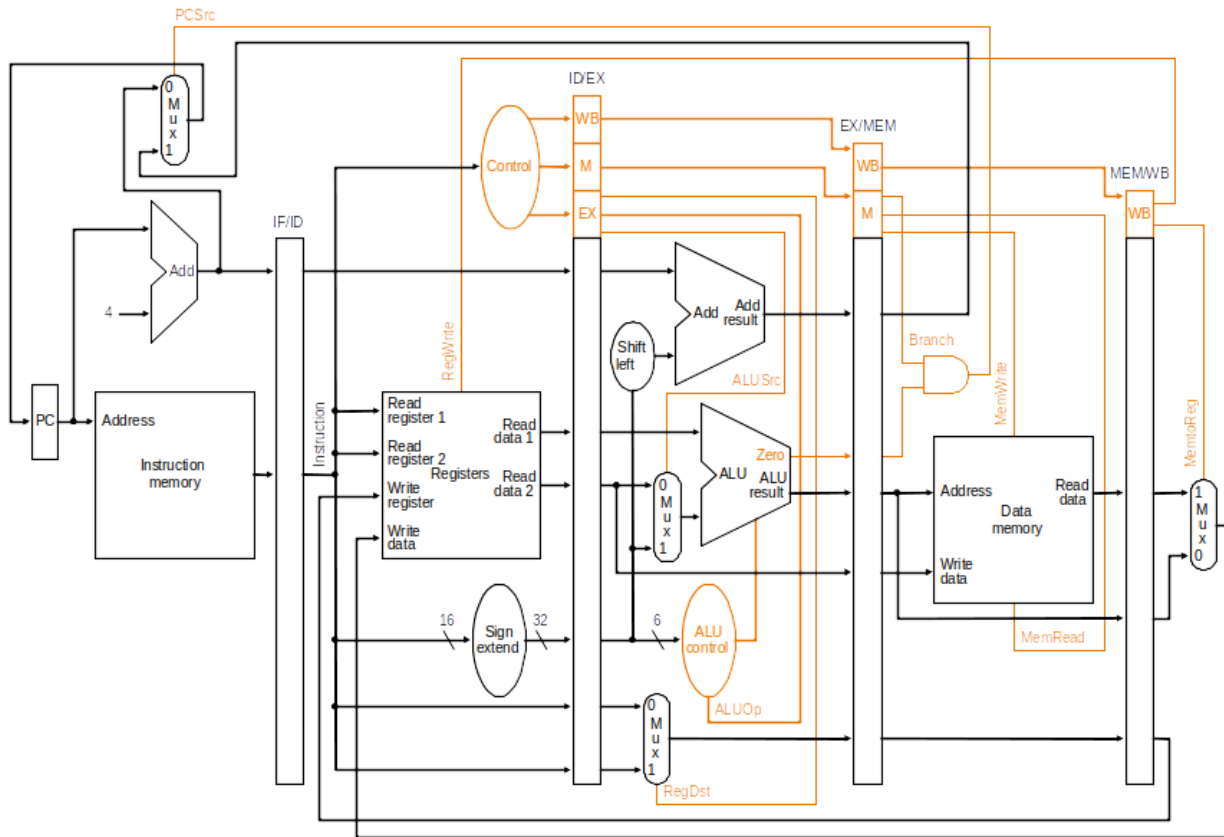
Controle do Pipeline



Controle do Pipeline



Controle do Pipeline



Pipeline ideal



- **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 35



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 cauda 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

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

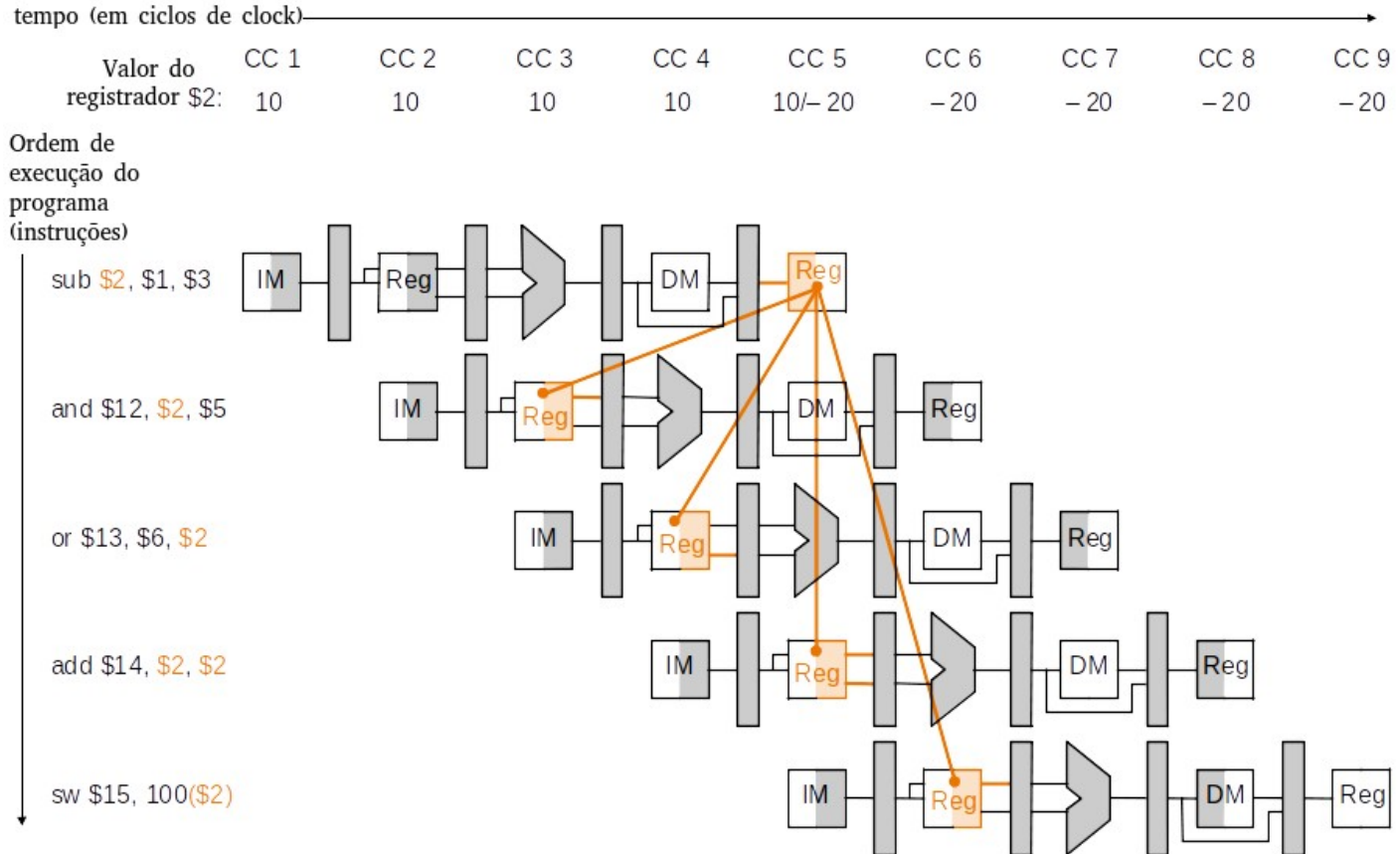
Write-after-Write
(WAW)

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



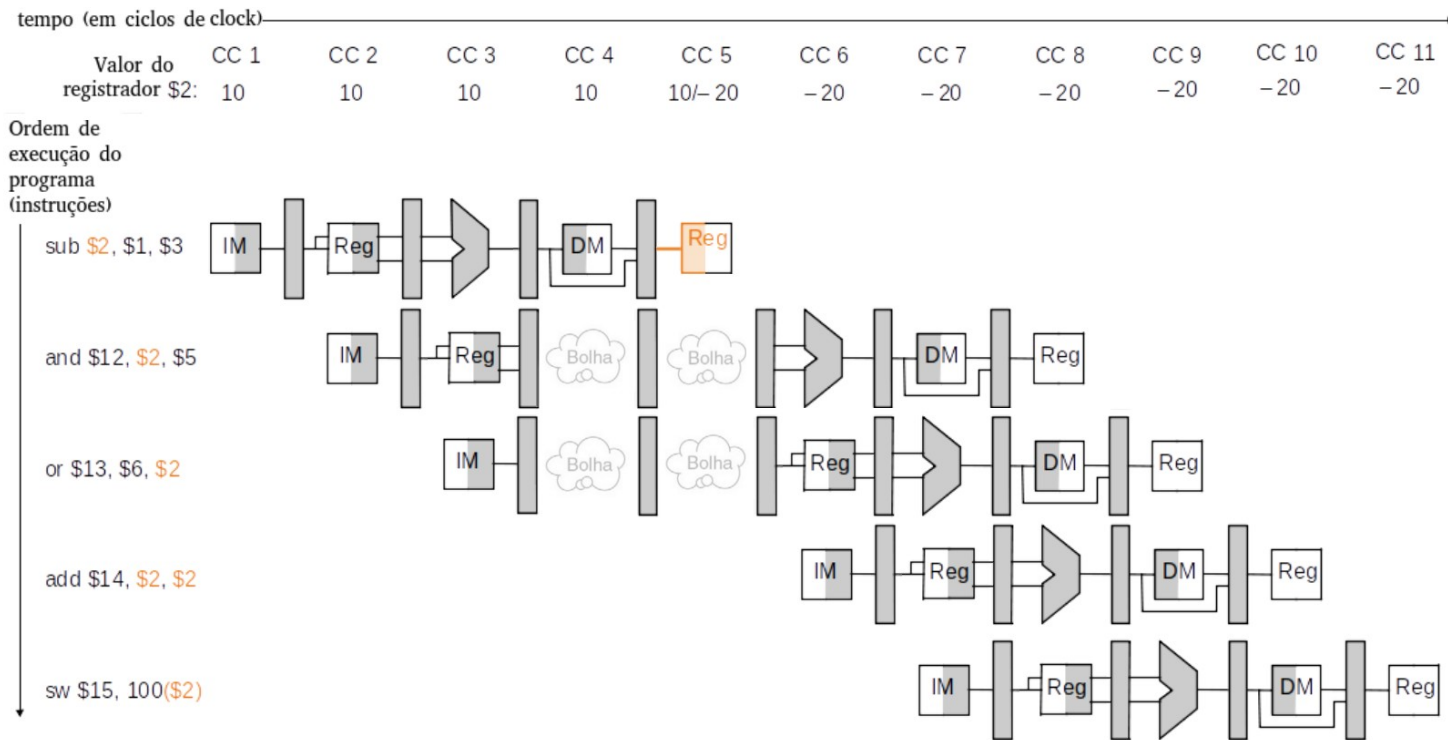
Pipeline: dependências de dados



- 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 quando a instrução no estágio ID deseja ler um registrador que ainda deve ser escrito e que se encontra nos estágios EX, MEM ou WB
- A instrução não pode avançar para o estágio EX e deve aguardar no estágio ID até que o registrador seja escrito no estágio WB

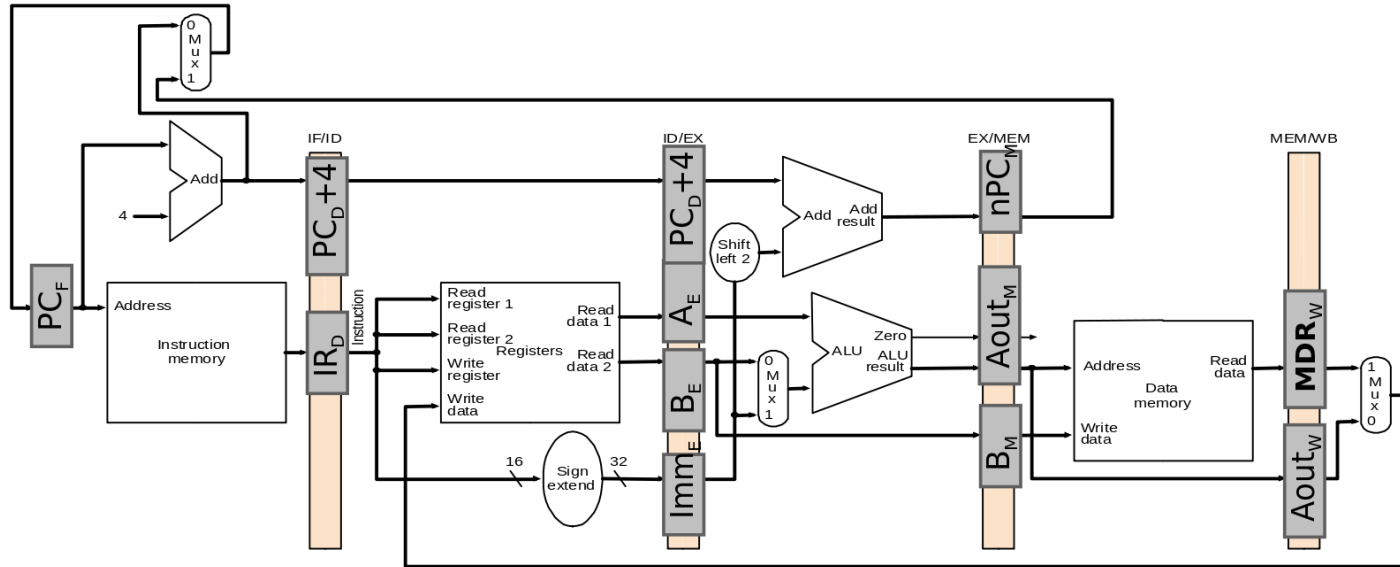
Pipeline: dependências de dados

1) Detectar e esperar o registrador ficar disponível

Como resolver?

Estágios anteriores:

- Desabilitar escrita em PC, IR e PC+4



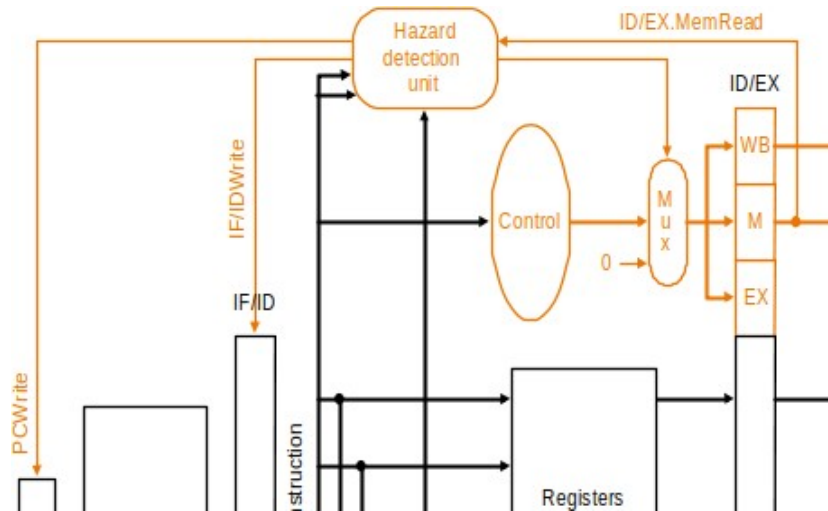
Pipeline: dependências de dados

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

Estágios: IF, ID, EX, MA, WB + NOP



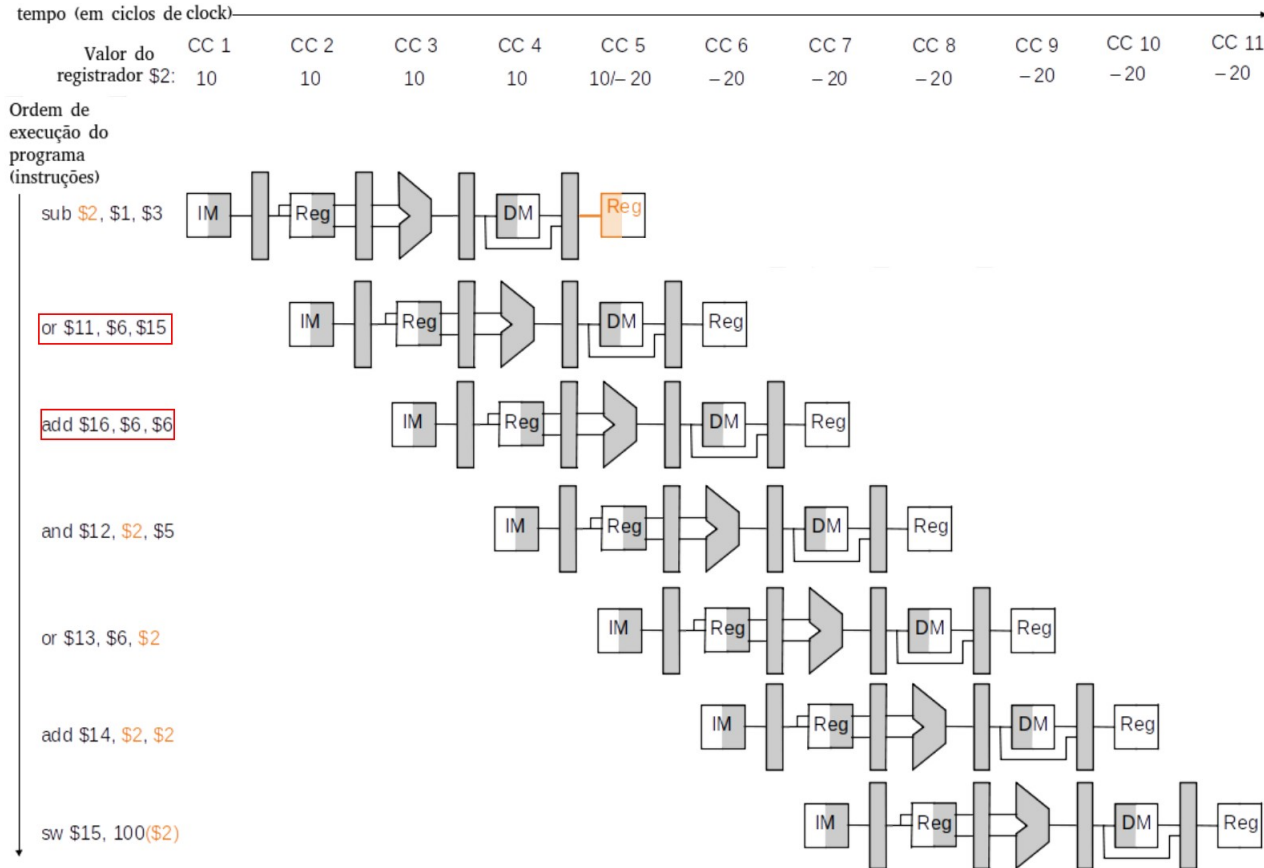
Exemplo:

```
.text
main:
    la    s0, VA
    li    t0, 0
    li    t1, 10
loop:
    beq   t0,t1, end
    lw    s1, 0(s0)
    addi  s1, s1,1
    sw    s1, 0(s0)
    addi  s0, s0, 4
    addi  t0, t0,1
    j     loop
end:
```

	CC1	CC2	CC3	CC4	CC5	CC6	CC7	CC8	CC9	CC10	CC11
la											
li											
li											
beq											
lw											
addi											
sw											
addi											
addi											
beq											

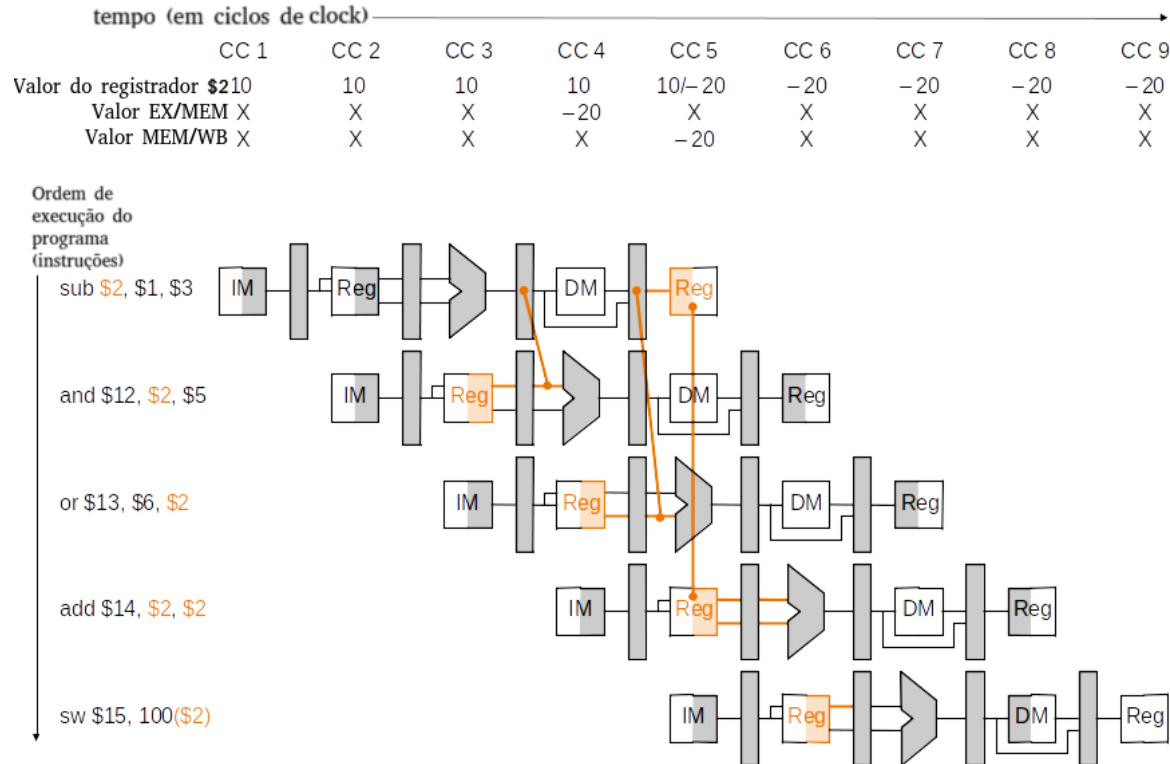
Pipeline: dependências de dados

2) Detectar e eliminar a dependência no nível de software



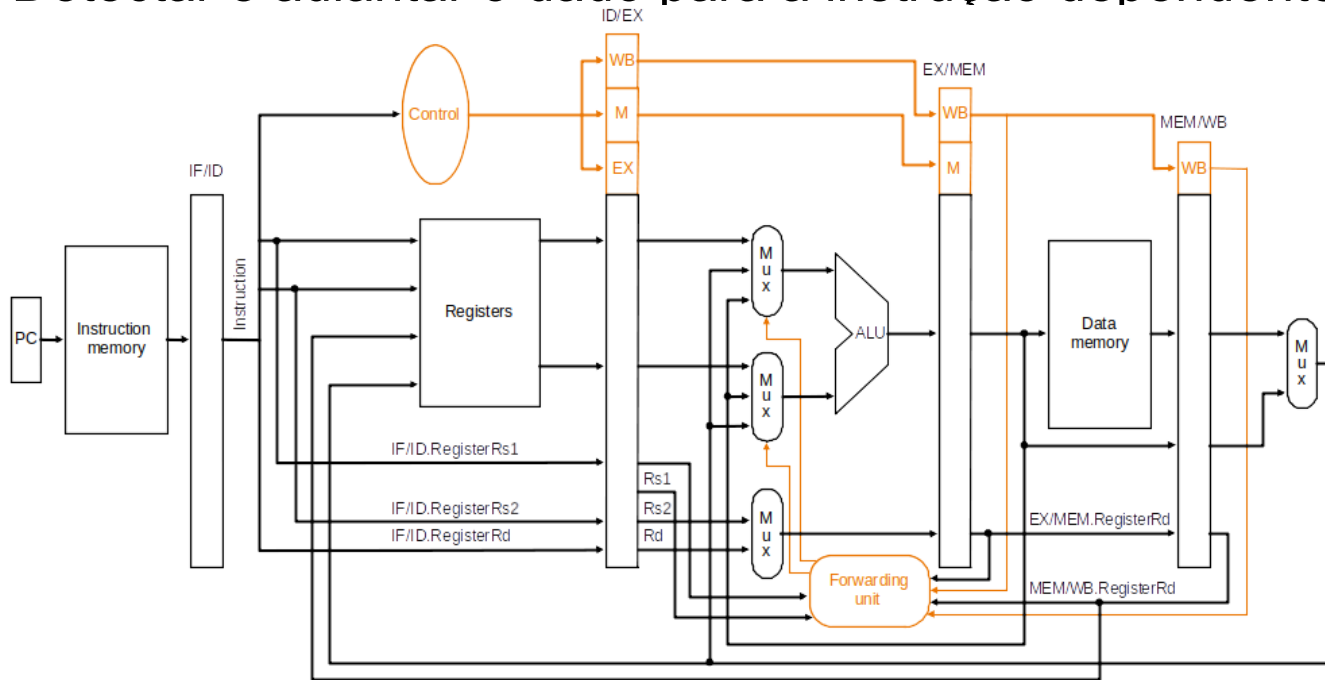
Pipeline: dependências de dados

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



Pipeline: dependências de dados

3) Detectar e adiantar o dado para a instrução dependente - forwarding



Pipeline

Estágios: IF, ID, EX, MA, WB (e FW)



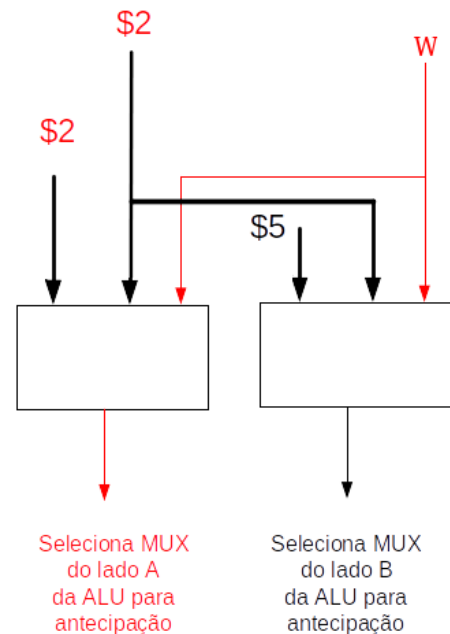
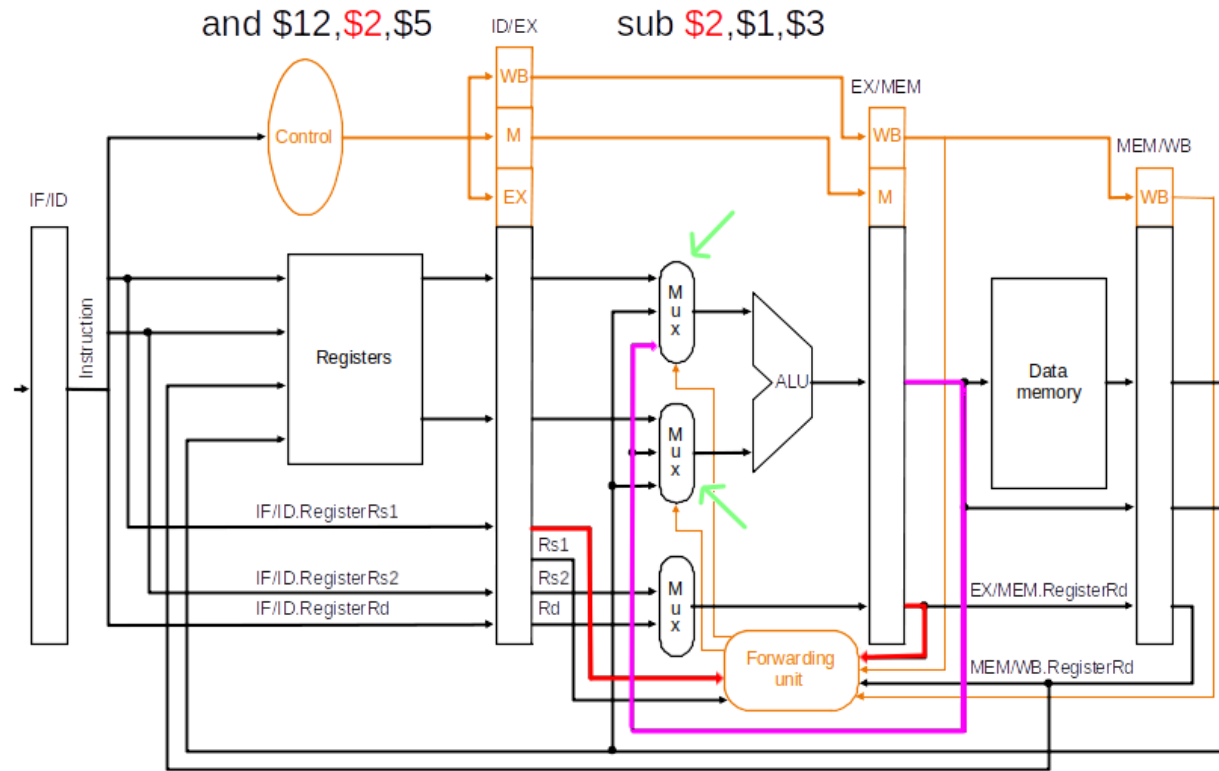
Exemplo:

```
.text
main:
    la    s0, VA
    li    t0, 0
    li    t1, 10
loop:
    beq   t0,t1, end
    lw    s1, 0(s0)
    addi  s1, s1,1
    sw    s1, 0(s0)
    addi  s0, s0, 4
    addi  t0, t0,1
    j     loop
end:
```

	CC1	CC2	CC3	CC4	CC5	CC6	CC7	CC8	CC9	CC10	CC11
la											
li											
li											
beq											
lw											
addi											
sw											
addi											
addi											
beq											

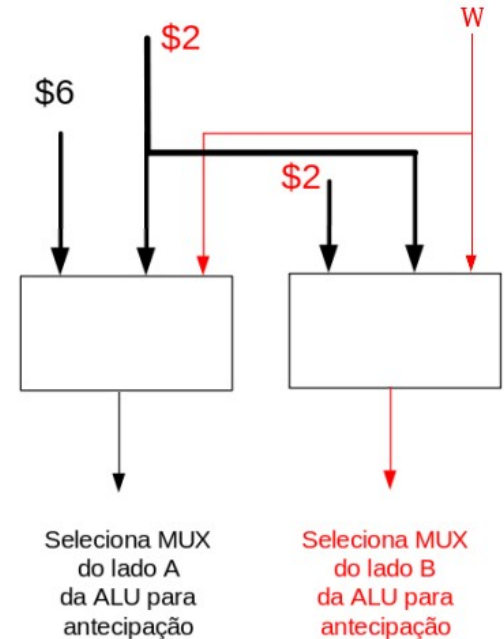
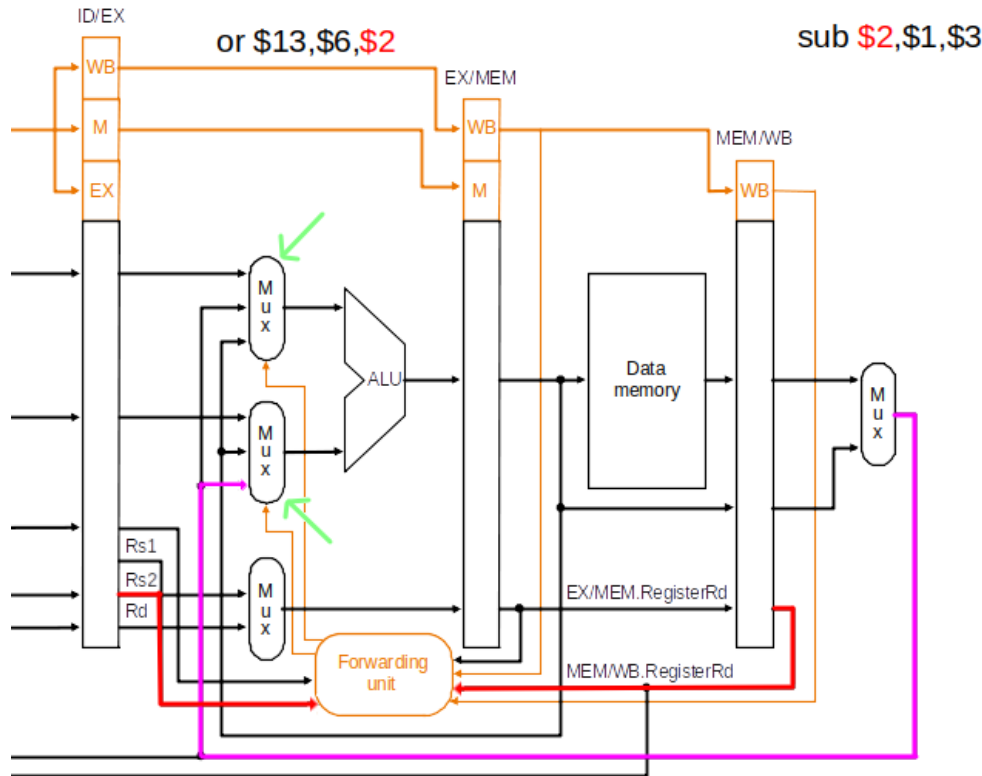
Pipeline: dependências de dados

3) Detectar e adiantar o dado para a instrução dependente - forwarding



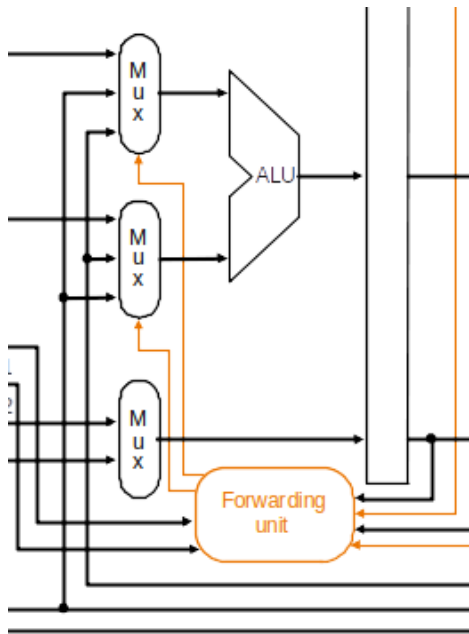
Pipeline: dependências de dados

3) Detectar e adiantar o dado para a instrução dependente - forwarding

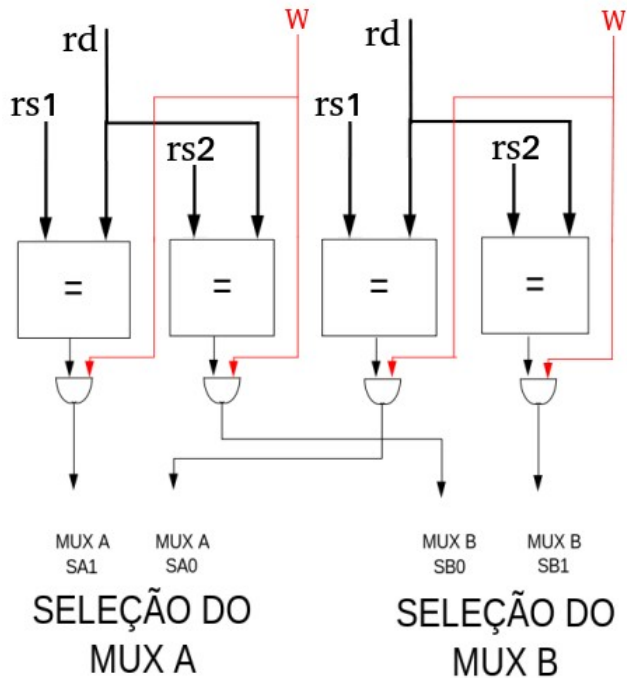


Pipeline: dependências de dados

3) Detectar e adiantar o dado para a instrução dependente - forwarding

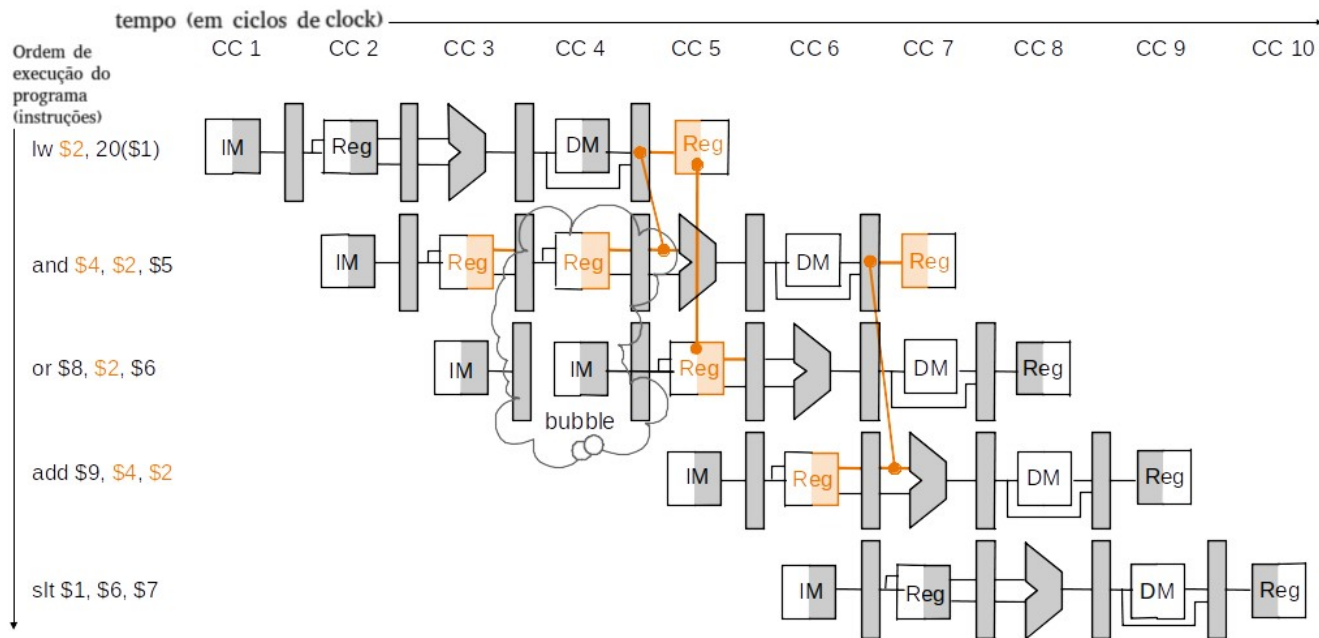


(estágio EX/MEM) (estágio MEM/WB)



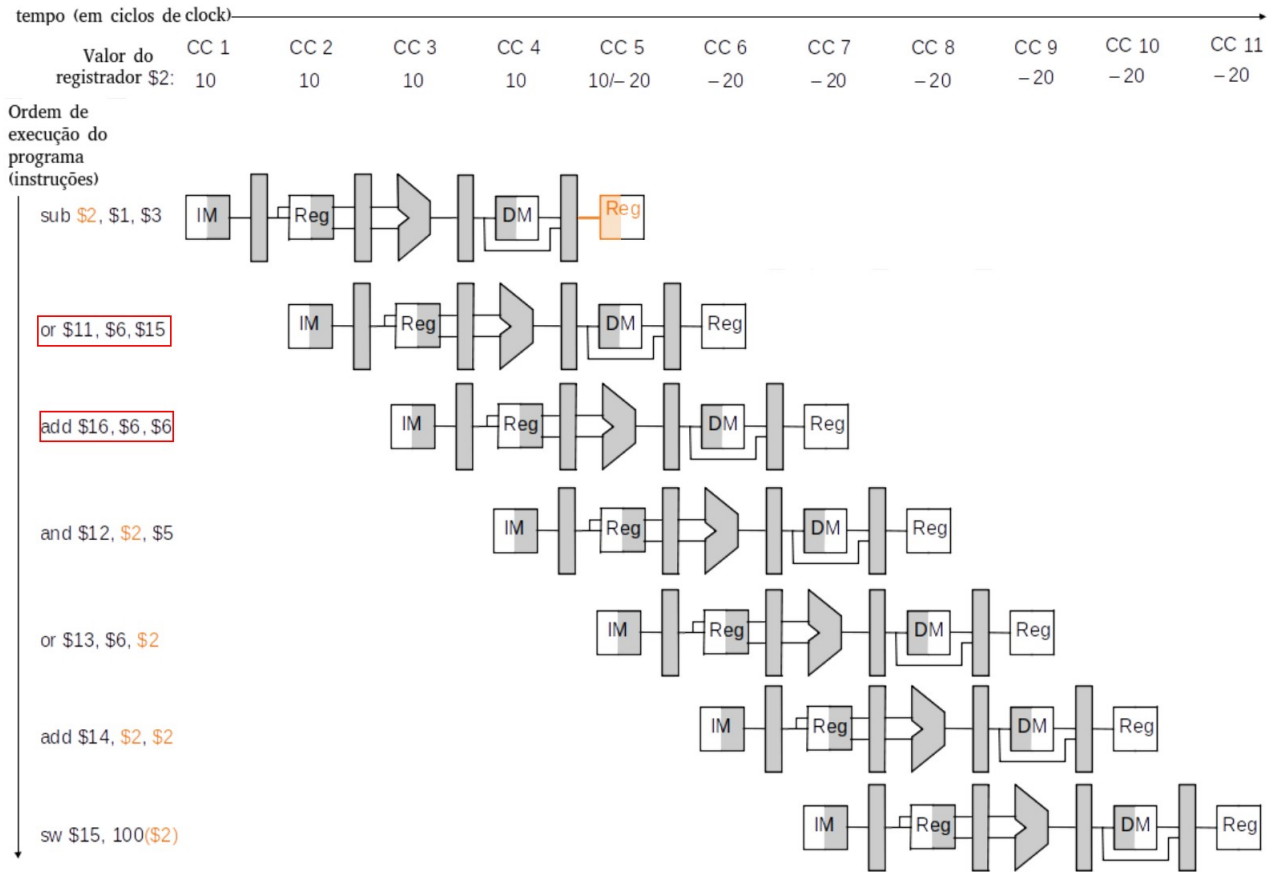
Pipeline: dependências de dados

- 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

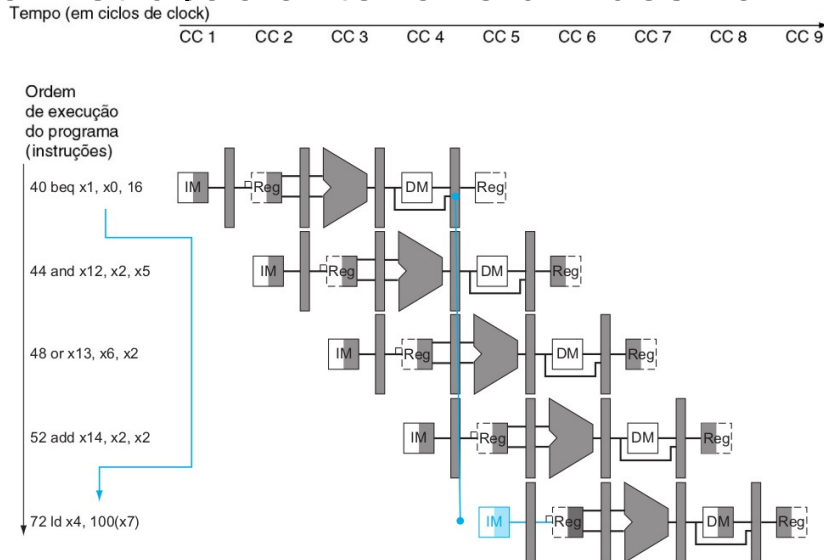


- **Pergunta:** Qual deve ser a próxima instrução a ser buscada?
Resposta: A instrução que estiver no endereço indicado pelo PC
- Todas as instruções a serem executadas, em algum sentido, dependem da instrução anterior
 - Se a instrução estiver no endereço de memória seguinte tudo ocorre maravilhosamente 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)

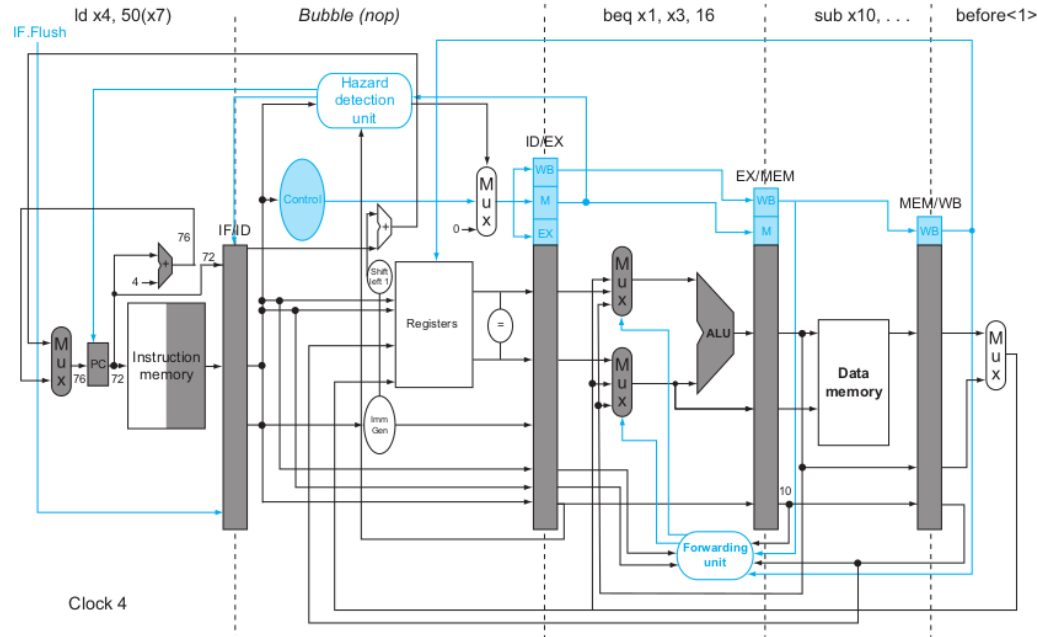
Pipeline: dependências de controle



- 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)

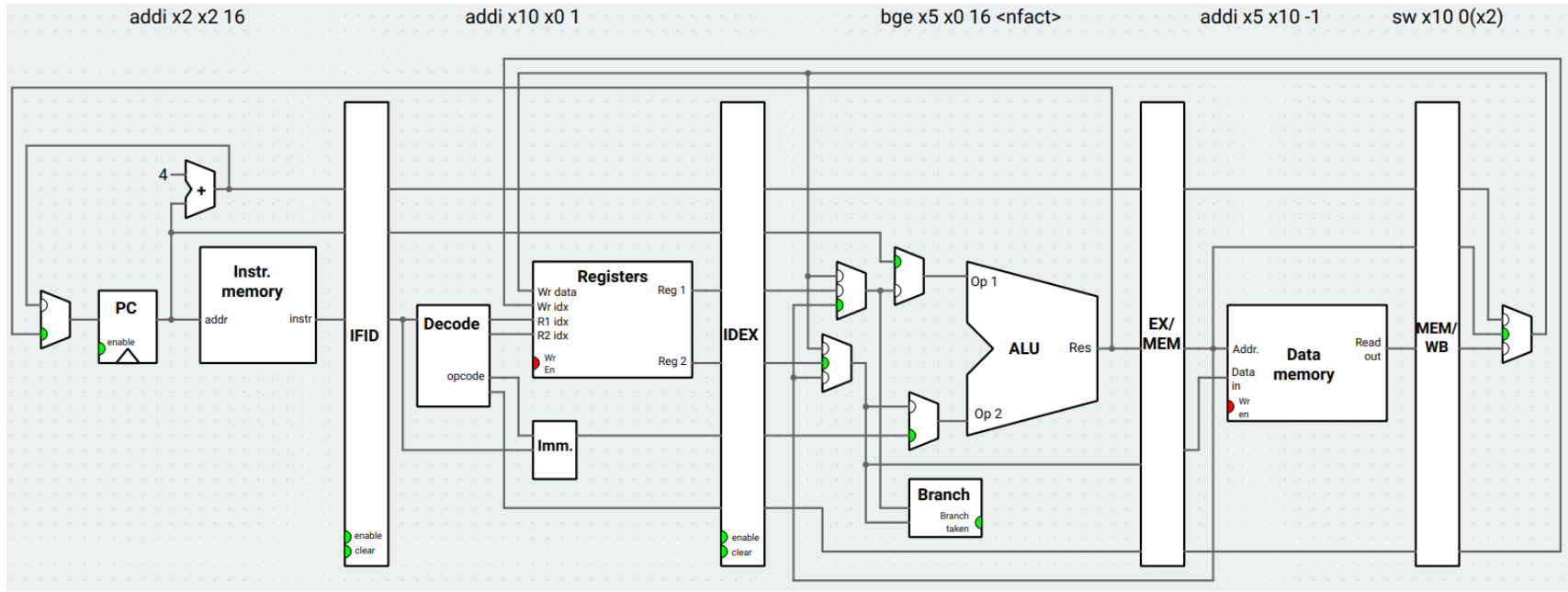
Pipeline: dependências de controle

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



Pipeline: dependências de controle

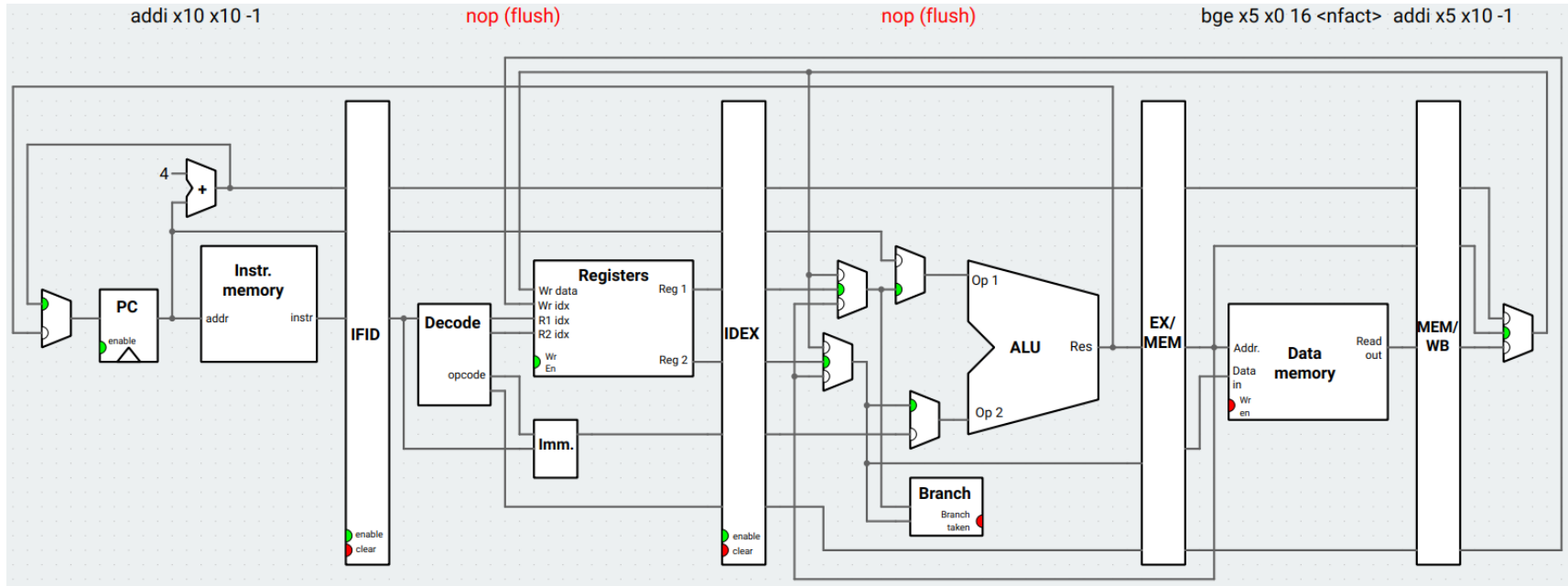
- a. Inserir bolhas (stalls) no pipeline até conhecer o endereço da próxima instrução a ser executada (flush instructions)



Pipeline: dependências de controle



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



Pipeline

Estágios: IF, ID, EX, MA, WB (a) WFW + FLUSH
(b) FW + FLUSH



Exemplo:

```
.text
main:
    addi t0, zero, 2
    addi t1, zero, 1
    add  s1, zero, zero
if:
    blt s0, t0, true
    addi s1, zero, -1
    j    fim
true:
    add s1, s1, t1
    addi t0, zero, 2
fim:
    add t0, zero, zero
```

	CC1	CC2	CC3	CC4	CC5	CC6	CC7	CC8	CC9	CC10	CC11
addi											
addi											
add											
blt											
addi											
j											
add											

Pipeline

Estágios: IF, ID, EX, MA, WB (+ FW + FLUSH)



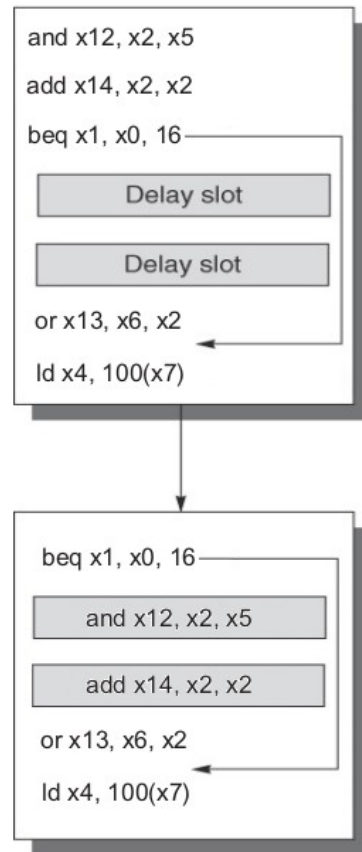
Exemplo:

```
.text
main:
    addi t0, zero, 2
    addi t1, zero, 1
    add  s1, zero, zero
if:
    blt  s0, t0, true
    addi s1, zero, -1
    j    fim
true:
    add  s1, s1, t1
    addi t0, zero, 2
fim:
    add  t0, zero, zero
```

	CC1	CC2	CC3	CC4	CC5	CC6	CC7	CC8	CC9	CC10	CC11
addi	IF	ID	EX	M	WB						
addi		IF	ID	EX	M	WB					
add			IF	ID	EX						
blt				IF	ID						
addi					IF						
j											
add											

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) →

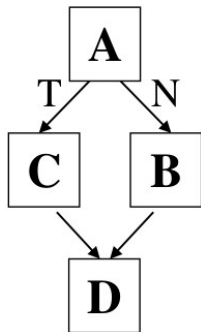


Pipeline: dependências de controle

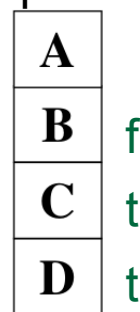


- c. Utilizar execução condicional (**predicated execution**)
- Idéia: Compilador converte dependência de controle em dependência de dado
 - Suporte no ISA pois instruções possuem um bit (predicado) indicando se a mesma será 'entregue' (committed); Exemplo ISA ARM
 - Somente instruções com predicado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

desvio normal



desvio predicado



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)

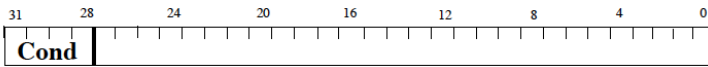
Swap

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.

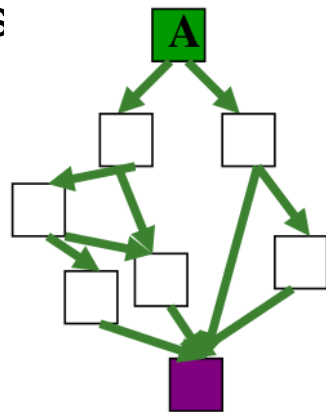
Pipeline: dependências de controle



- 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

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

mantendo a busca de instruções funcionando e o pipeline cheio

É uma boa previsão?

Expectativa: $\text{'nextPC} = \text{PC} + 4\text{'}$ ~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



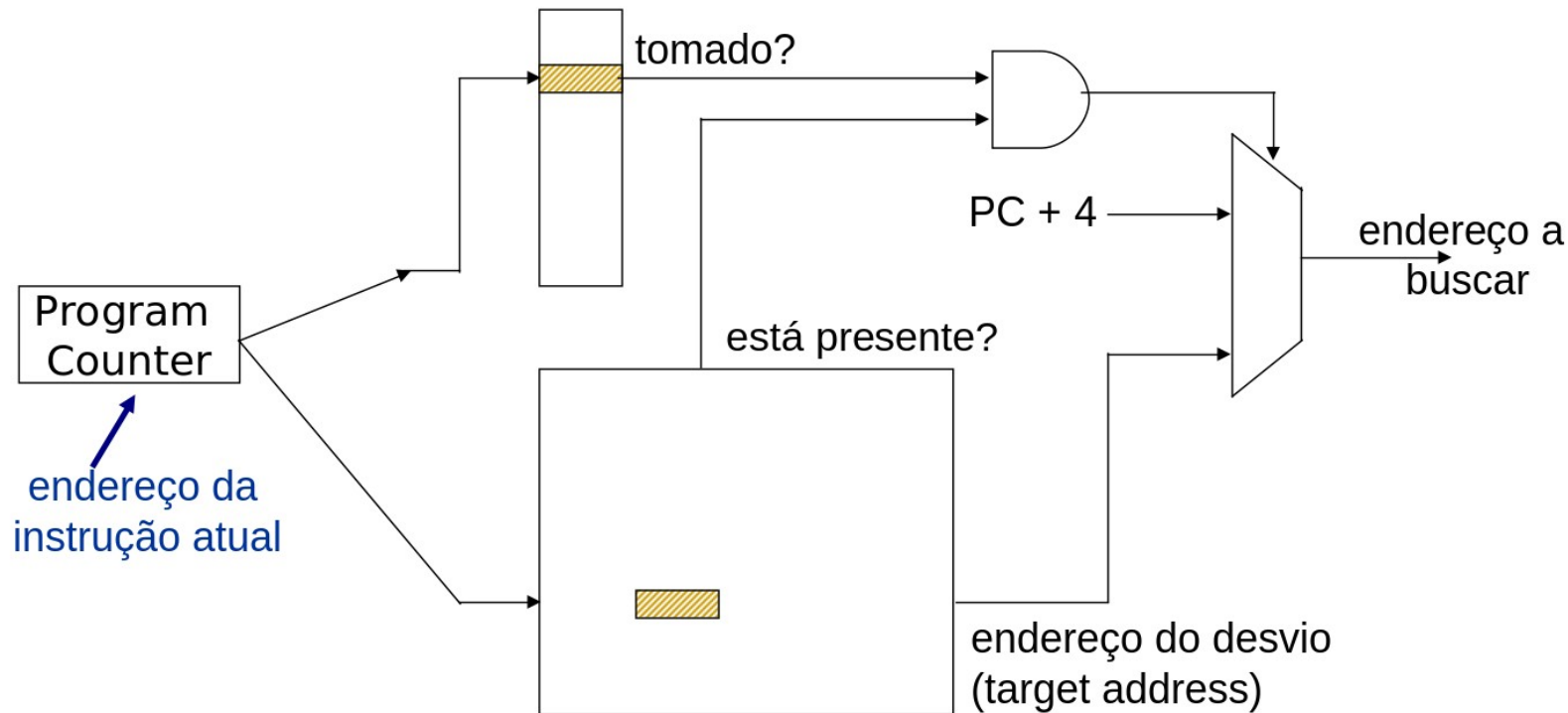
d. Utilizar predição de desvio

- 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



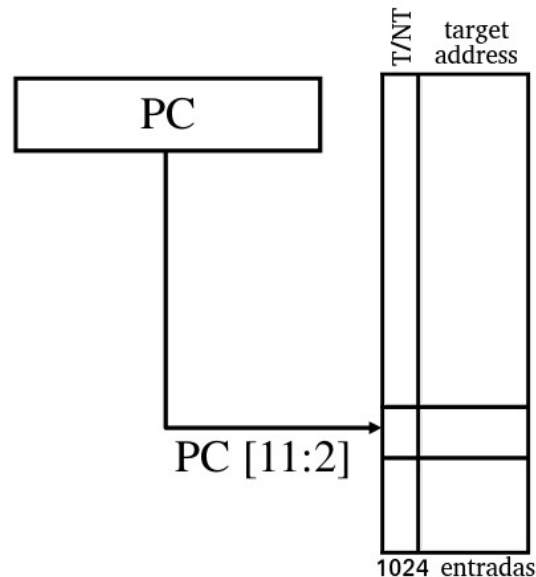
d. Utilizar predição de desvio

- 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)

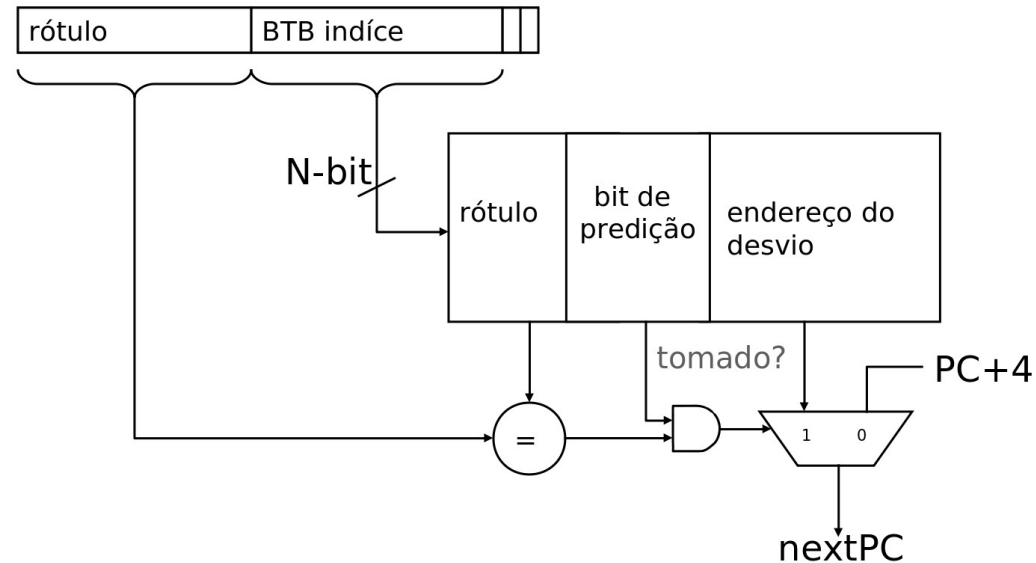
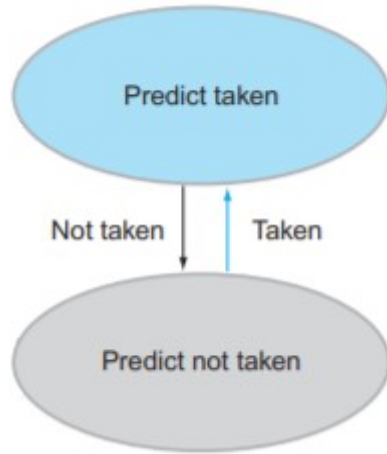


Pipeline: dependências de controle

d. Utilizar previsão de desvio

- Previsão 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

