MC542

Organização de Computadores
Teoria e Prática

2006

Prof. Paulo Cesar Centoducatte

ducatte@ic.unicamp.br

www.ic.unicamp.br/~ducatte

MC542

Arquitetura de Computadores

Processador MIPS
Pipeline

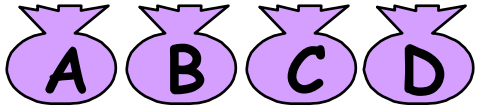
"Computer Organization and Design:
The Hardware/Software Interface" (Capítulo 6)

Sumário

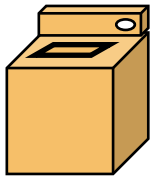
- Conceito de Pipeline
  - Exemplo
- Pipeline do MIPS
  - Um RISC Típico
- Pipeline de instruções no MIPS
- Execução Não-Pipeline X Pipeline
- Limites de Pipelining
  - Hazards
    - Dados
    - Controle
    - Estrutural

Pipelining - Conceito

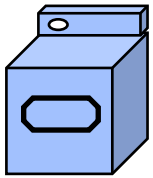
- **Exemplo: Lavanderia**
- 4 trouxas para serem lavadas



  - Lavar: 30 minutes

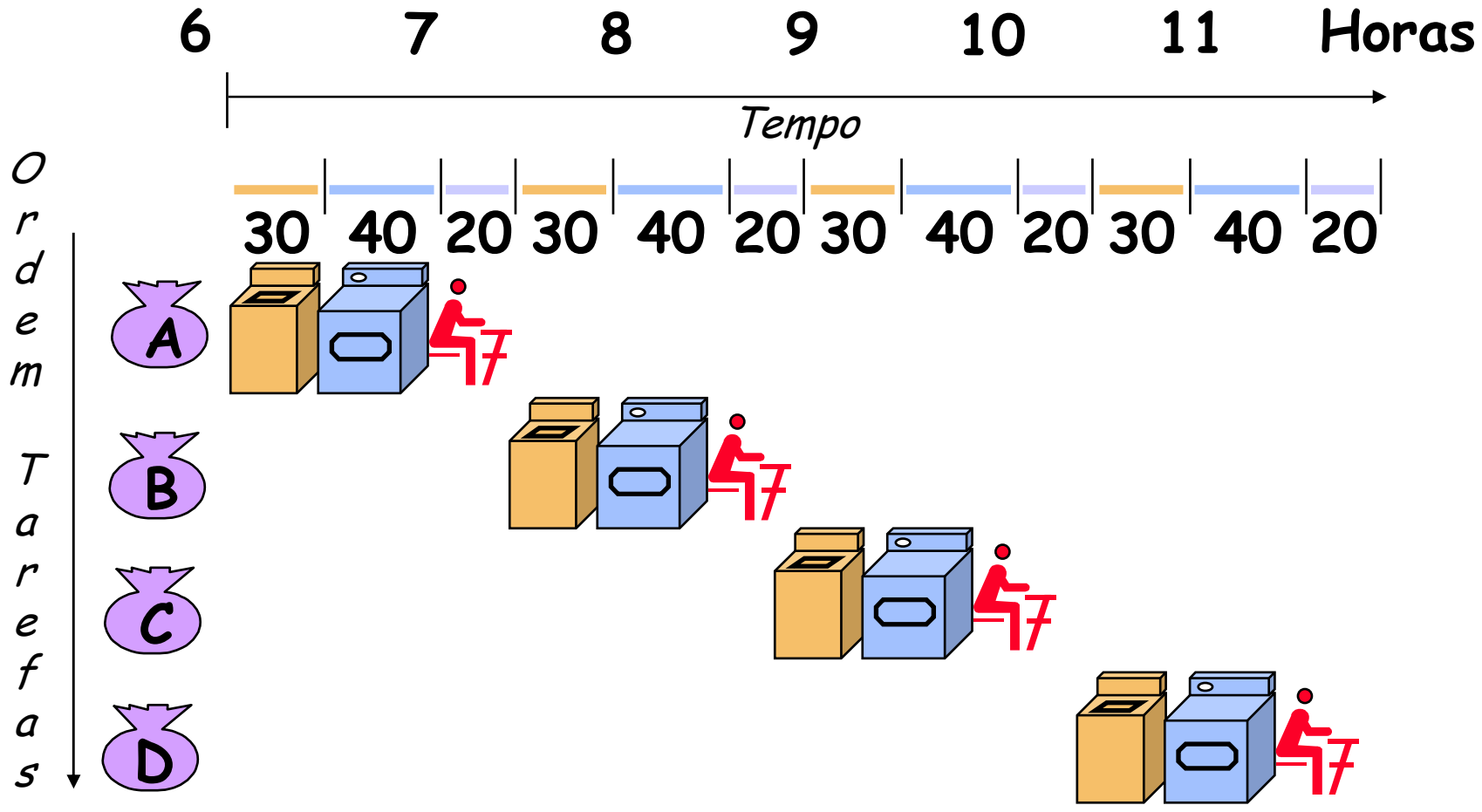


  - Secar: 40 minutes



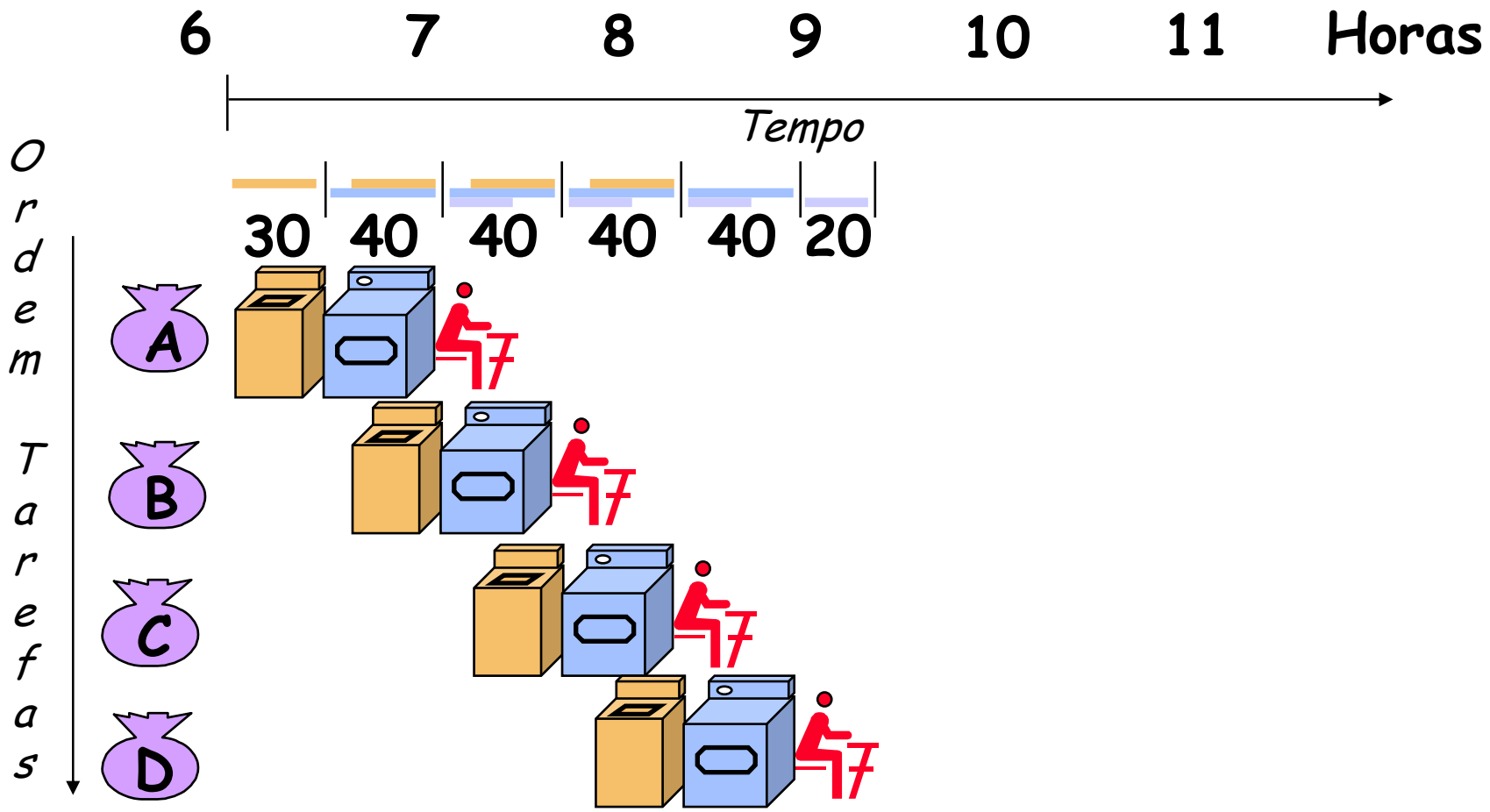
  - Passar: 20 minutes

Lavanderia Seqüencial



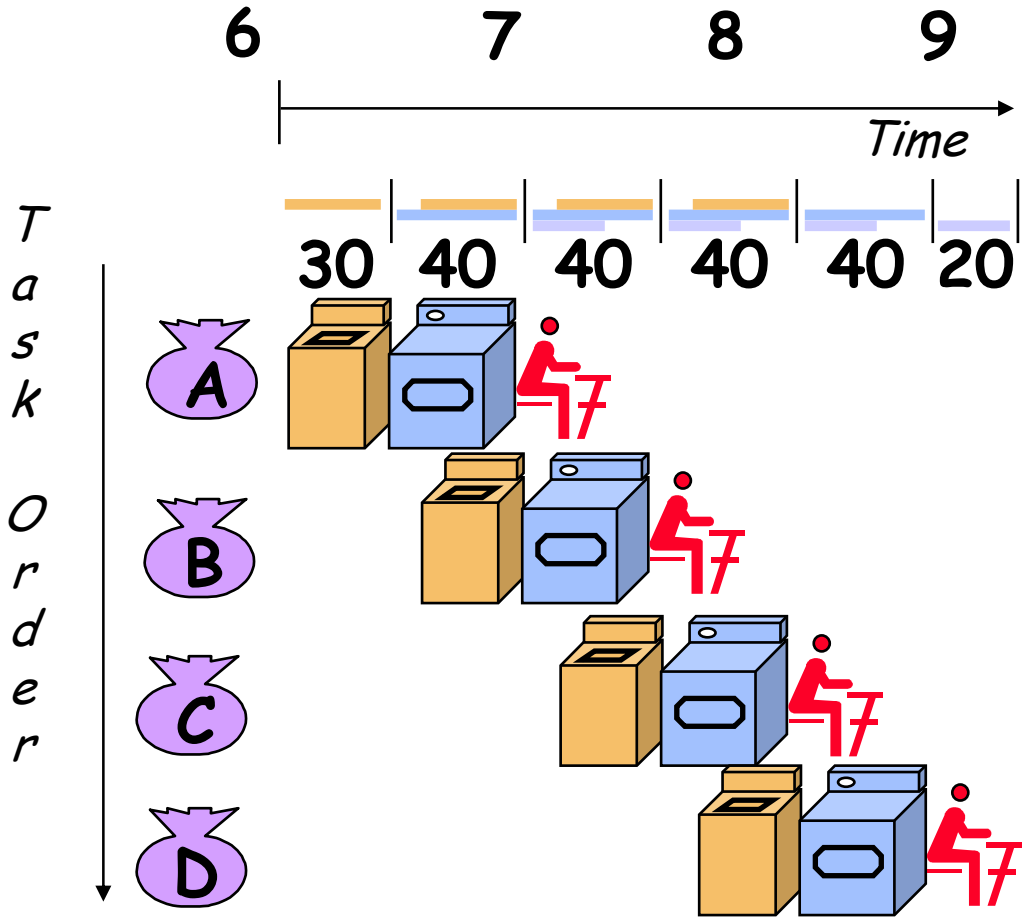
- Lavanderia seqüencial: 6 horas para lavar 4 trouxas
- Se for usado pipelining, quanto tempo?

Lavanderia Pipelined



- Lavanderia Pipelined: 3.5 horas para 4 trouxas

Pipelining



- O Pipelining não ajuda na latência de uma tarefa, ajuda no throughput de toda a carga de trabalho
- O período do Pipeline é limitado pelo estágio mais lento
- Multiplas tarefas simultâneas
- Speedup potencial = Número de estágios
- Estágios não balanceados reduzem o speedup
- O Tempo de "preenchimento" e de "esvaziamento" reduzem o speedup

CPU Pipelines

- Executam bilhões de instruções: *throughput*

- Características desejáveis em um conjunto de instruções (ISA) para pipelining?

  - Instruções de tamanho variável vs. Todas instruções do mesmo tamanho?
  - Operandos em memória em qq operações vs. operandos em memória somente para loads e stores?
  - Formato das instruções irregular vs. formato regular das instruções (ié. Operandos nos mesmos lugares)?

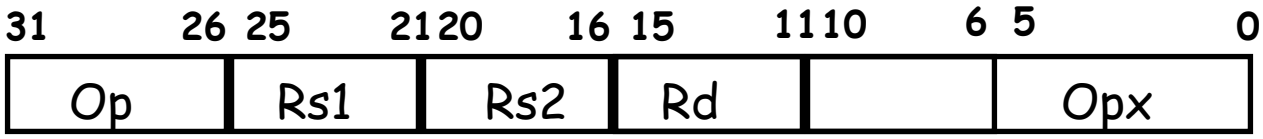
Um RISC Típico

- Formato de instruções de 32-bit (3 formatos)
- Acesso à memória somente via instruções load/store
- 32 32-bit GPR (R0 contains zero)
- Instruções aritméticas: 3-address, reg-reg, registradores no mesmo lugar
- Modo de endereçamento simples para load/store (base + displacement)
  - Sem indireção
- Condições simples de branch
- Delayed branch

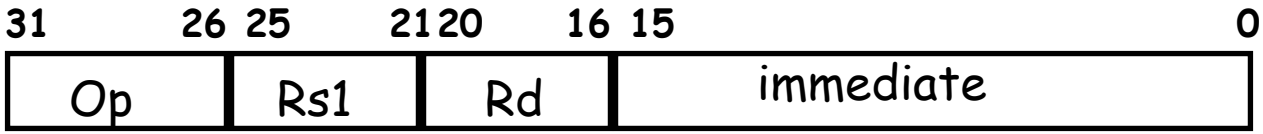
SPARC, MIPS, HP PA-Risc, DEC Alpha, IBM PowerPC, CDC 6600, CDC 7600, Cray-1, Cray-2, Cray-3

Exemplo: MIPS (Localização dos regs)

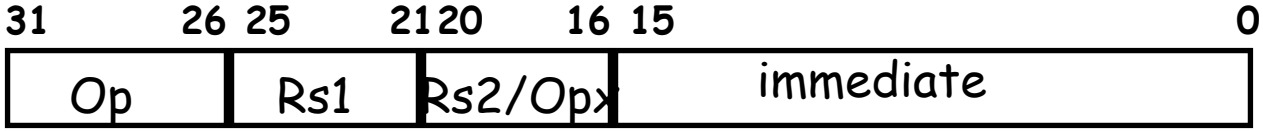
Register-Register



Register-Immediate



Branch



Jump / Call

Pipeline de instruções no MIPS

- Fetch da instrução
- Leitura dos registradores e decodificação
- Execução da operação ou cálculo de endereço
- Acesso ao operando na memória
- Escrita do resultado em um registrador

Exemplo

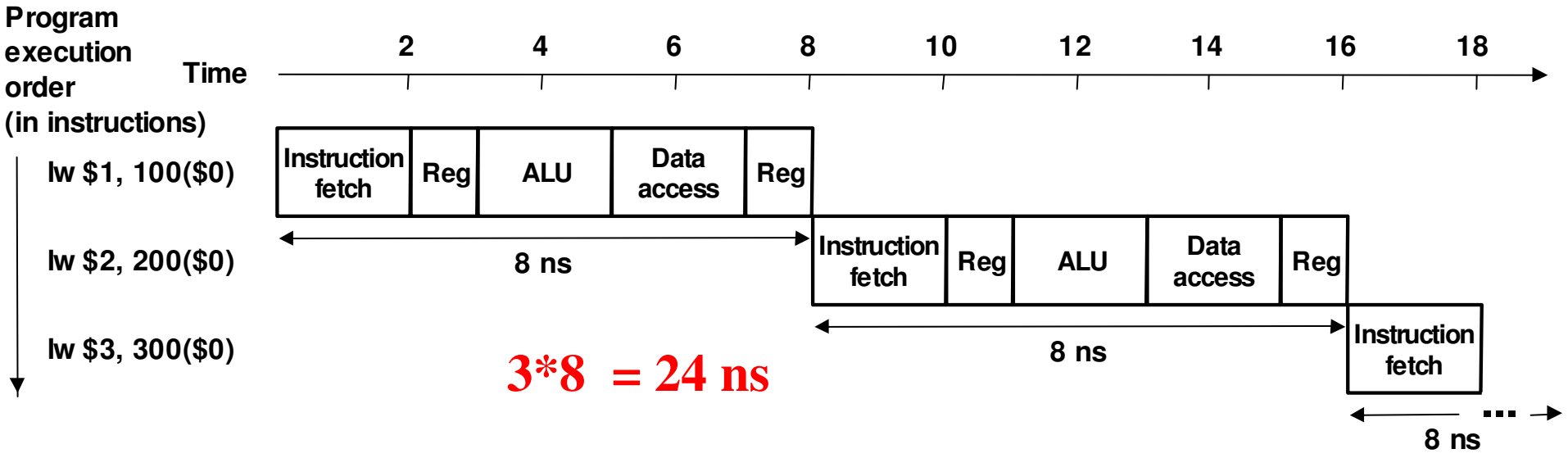
- Compare o tempo médio entre instruções da implementação em single-cycle (uma instrução por ciclo) com uma implementação com pipeline. Supor maior tempo de operação para acesso à memória = 2ns, operação da ULA = 2ns e acesso ao register file = 1ns. (Instrs lw, sw, add, sub, and, or slt e beq).

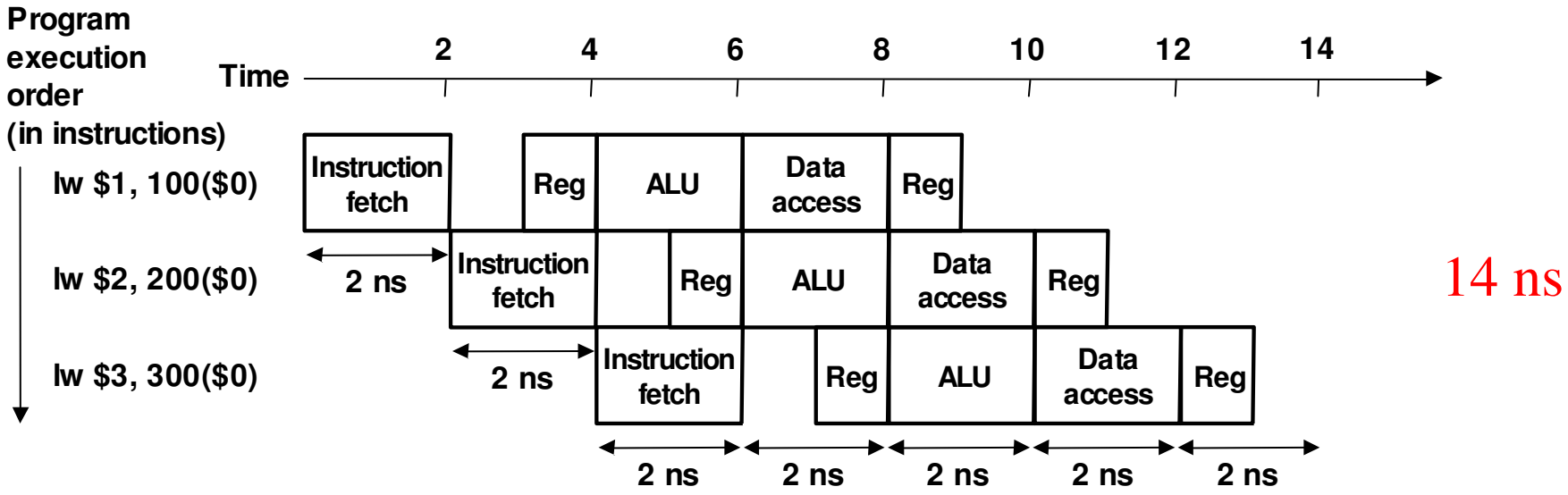
- Inicialmente suponha a execução de 3 instruções lw (tempo entre o início da 1ª instrução e o início da 4ª instrução)

Tempo Total para as Oito Instruções Calculado a Partir do Tempo de cada Componente

Instruction class	Instruction fetch	Register read	ALU operation	Data access	Register write	Total time
Load word (lw)	2 ns	1 ns	2 ns	2 ns	1 ns	8 ns
Store word (sw)	2 ns	1 ns	2 ns	2 ns		7 ns
R-format (add, sub, and, or, slt)	2 ns	1 ns	2 ns		1 ns	6 ns
Branch (beq)	2 ns	1 ns	2 ns			5 ns

Execução Não-Pipeline X Pipeline





OBS.:

- Sob condições ideais, com estágios balanceados, o speedup do pipeline é igual ao número de estágios do pipeline (5 estágios , 5 vezes mais rápido)
- Na realidade o tempo de execução de uma instrução é um pouco superior (overheads) ➔ speedup é menor que o número de estágios do pipeline

Desempenho do pipeline é devido ao aumento do throughput.

Suponha a execução de 1003 instruções

com pipeline ➔

1000 X 2ns + 14 = 2014 (para cada instrução adiciono 2ns)

sem pipeline ➔ 1000 X 8ns + 24 = 8024

spedup = 8024 / 2014 = 3.98 ~~ 8 / 2

Limites de Pipelining

- **Hazards:** impedem que a próxima instrução seja executada no ciclo de clock "previsto" para ela.
  - **Structural hazards**: O HW não suporta uma dada combinação de instruções
  - **Data hazards**: Uma Instrução depende do resultado da instrução anterior que ainda está no pipeline
  - **Control hazards**: Causado pelo delay entre o fetching de uma instrução e a decisão sobre a mudança do fluxo de execução (branches e jumps).

Projeto de um Conjunto de Instruções para Pipeline

- O que torna a implementação mais fácil?
  - Instruções de mesmo tamanho
  - Poucos formatos, com campos de registradores sempre dispostos no mesmo lugar (Simetria, no 2° estágio podemos ler registradores e decodificar ao mesmo tempo).
  - Acesso à memória apenas com as instruções lw e sw.
  - Operandos alinhados na memória: o dado pode ser
  - transferido da memória para a CPU e CPU para a memória em um único estágio do pipeline.

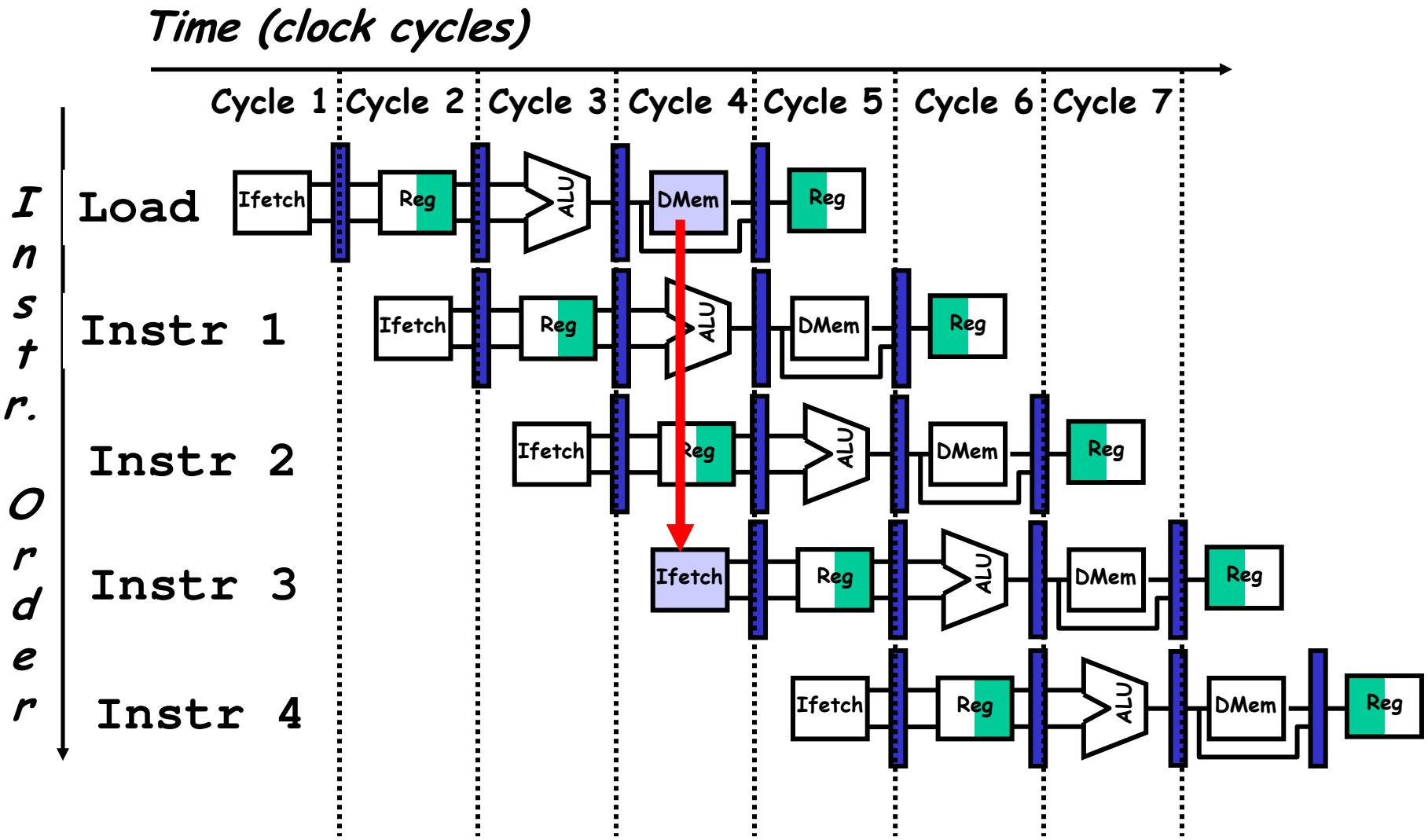
Projeto de um Conjunto de Instruções para Pipeline

- O que torna a implementação mais difícil?
  - Hazard
    - Hazard Estrural
    - Hazard de Controle
    - Hazard de Dados

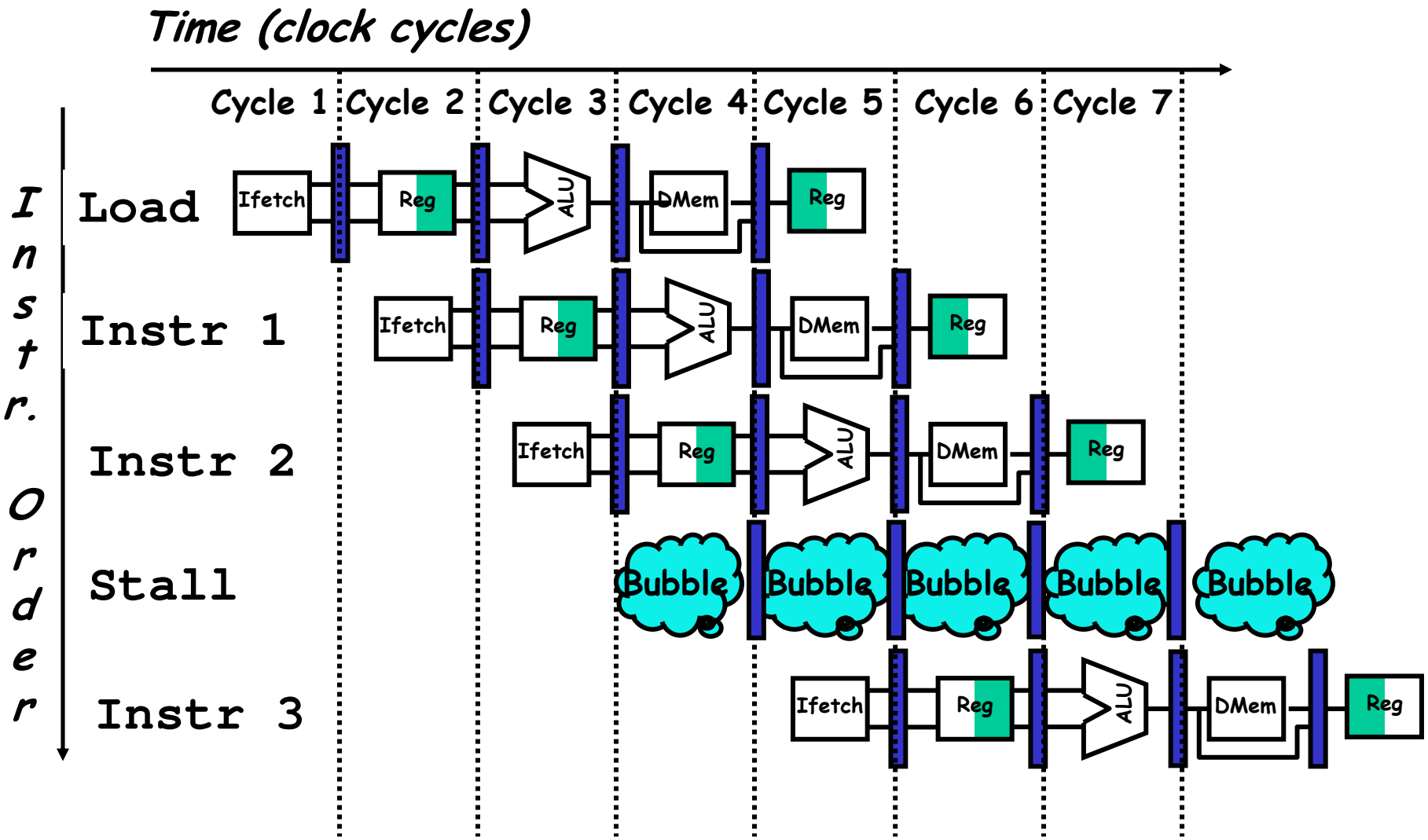
Pipeline Hazards

- **Hazard Estrutural**
  - **O hardware não suporta uma combinação de instruções que queremos executar em um único período de clock**
    - **Ex.: escrever e ler da memória em um mesmo ciclo**

Memória Única (D/I) - Structural Hazards



Memória única (D/I) - Structural Hazards

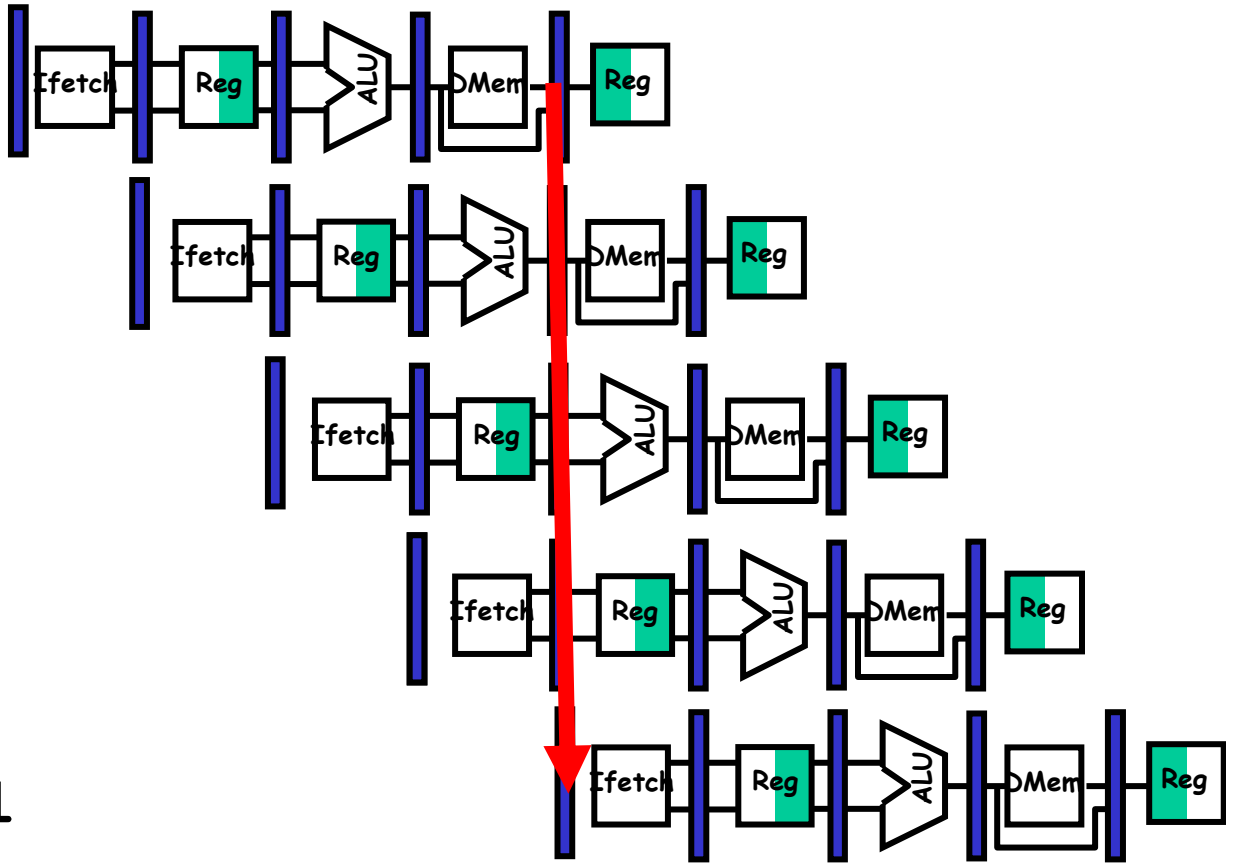


Pipeline Hazards

- **Hazard de Controle**
  - Problemas devido à execução de instruções de desvio
    - Ex.: Quando um branch é tomado, como tratar a(s) instruções que seguem (fisicamente) o branch no programa e que já estão no pipeline

Control Hazard - Branches (3 Estágios de Stall)

```
10: beq r1,r3,36
14: and r2,r3,r5
18: or  r6,r1,r7
22: add r8,r1,r9
36: xor r10,r1,r11
```



Alternativas para Branch Hazard

#1: Stall até a decisão se o branch será tomado ou não

#2: Predict Branch Not Taken

- Executar a próxima instrução
- "Invalidar" as instruções no pipeline se branch é tamado
- Vantagem: retarda a atualização do pipeline
- 47% dos branches no MIPS não são tomados, em média
- PC+4 já está computado, use-o para pegar a próxima instrução

#3: Predict Branch Taken

- 53% dos branches do MIPS são tomados, em média
- "branch target address" no MIPS ainda não foi calculado
  - 1 cycle branch penalty
  - Em outras máquinas esse penalty pode não ocorrer

Alternativas para Branch Hazard

#4: Delayed Branch

- Define-se que o branch será tamado *APÓS* a uma dada quantidade de instruções

```
branch instruction
  sequential successor1
  sequential successor2
  ........
  sequential successorn
branch target if taken
```

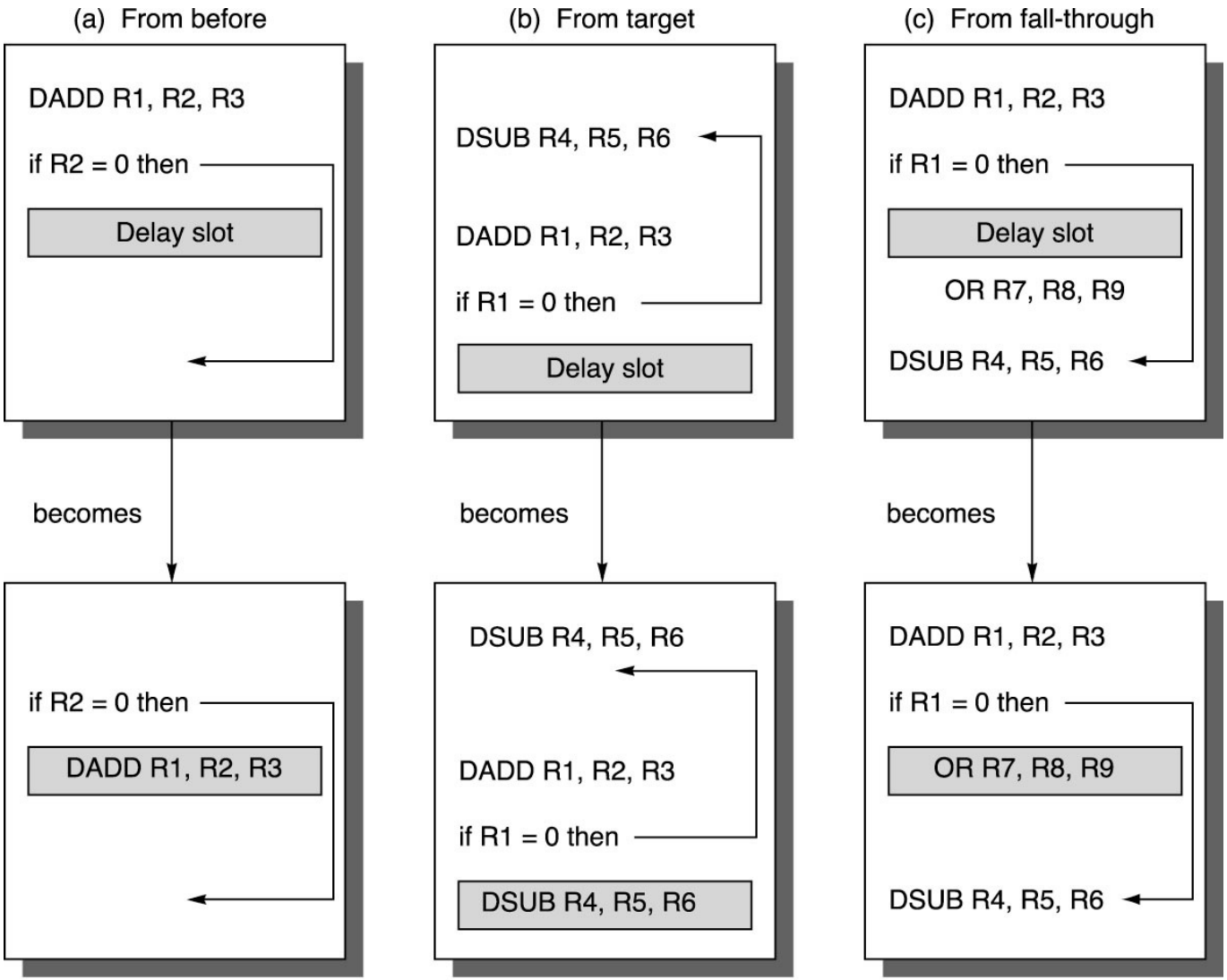
Branch delay de tamanho *n*
(n slots delay)

- 1 slot delay permite a decisão e o calculo do "branch target address" no pipeline de 5 estágios
- MIPS usa está solução

Delayed Branch

- Qual instrução usar para preencher o branch delay slot?
  - Antes do branch
  - Do target address (avaliada somente se branch taken)
  - Após ao branch (somente avaliada se branch not taken)

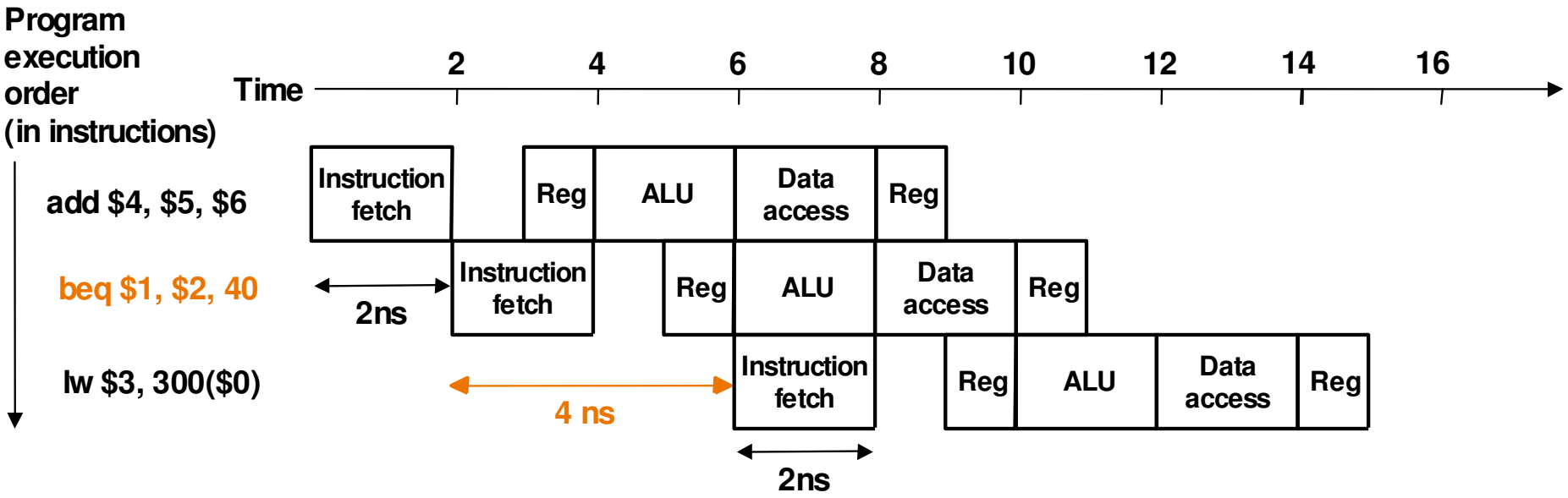
Delayed Branch



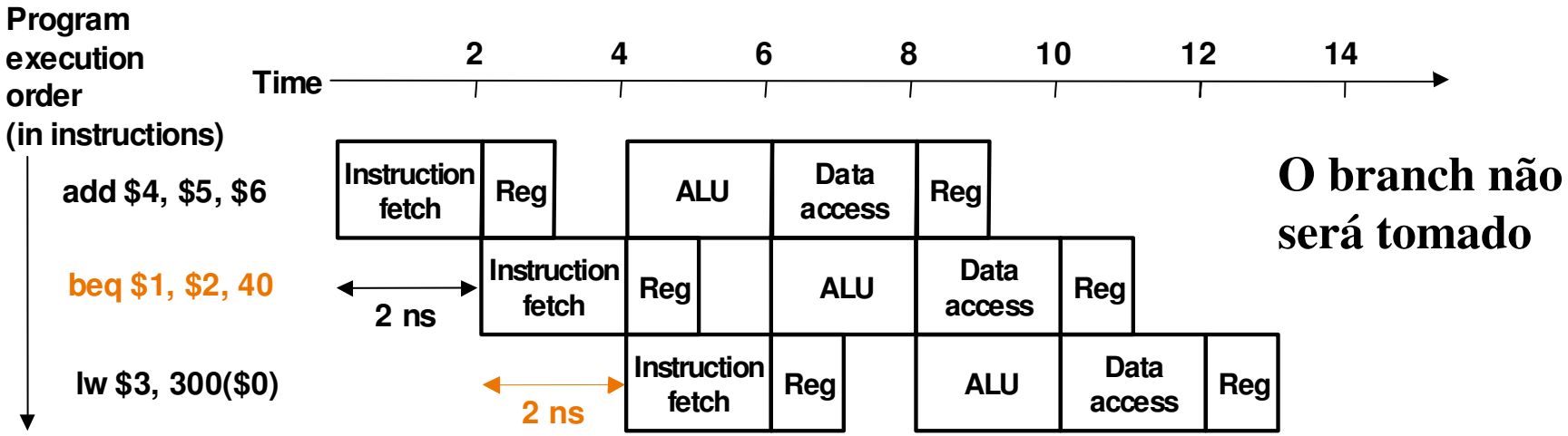
Delayed Branch

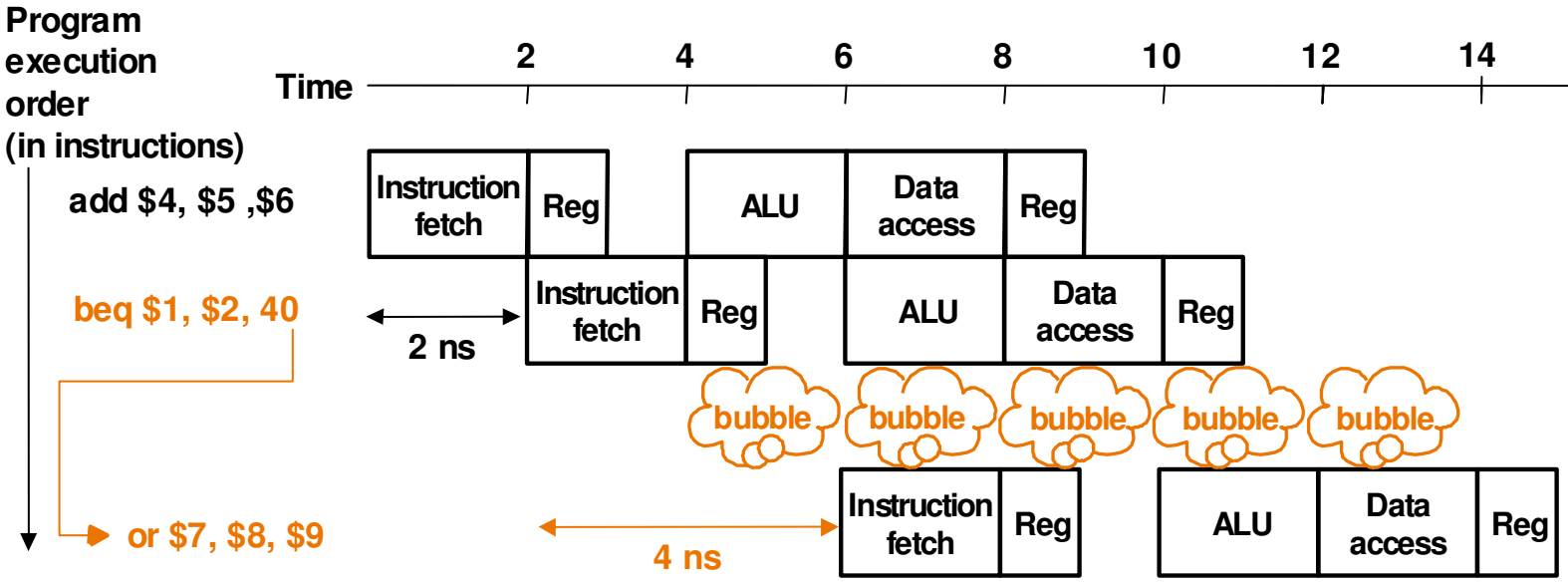
- Compilador: single branch delay slot:
  - Preenche +/- 60% dos branch delay slots
  - +/- 80% das instruções executadas no branch delay slots são utéis à computação
  - +/- 50% (60% x 80%) dos slots preencidos sáo utéis

Pipelining stalling para Instruções Branch

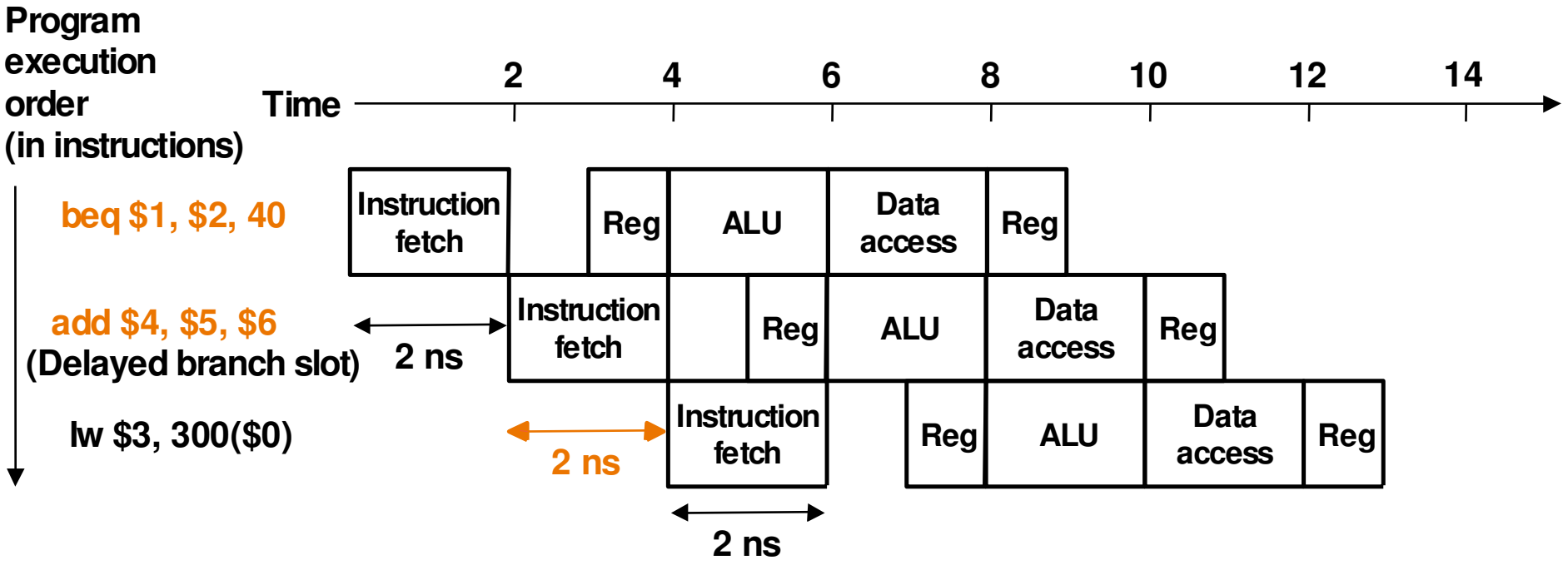


Branch prediction: Tentar "Adivinhar" qual dos Caminhos do Branch será Tomado





Pipeline Delayed Branch



Exemplo: Impacto do Branch Stall

- Se CPI = 1, 30% branches, 3-cycle stall

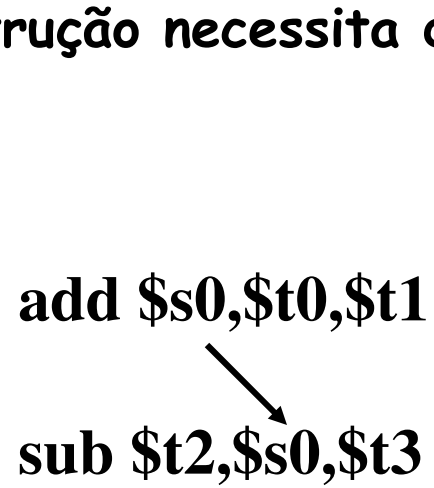
  $\Rightarrow$ **CPI = 1.9!**

- Solução para minimizar os efeitos:
  - Determinar branch taken ou não o mais cedo, e
  - Calcular o endereço alvo do branch logo
- MIPS branch: testa se reg = 0 ou ≠ 0
- Solução MIPS:
  - Zero test no estágio ID/RF
  - Adder para calcular o novo PC no estágio ID/RF
  - 1 clock cycle penalty por branch versus 3

Hazard de Dados

- Quando uma instrução necessita de um dado que ainda não foi calculado

  - Ex.:



Soluções :

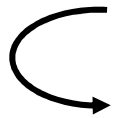
Compilador (programador) gera código livre de data hazard (introduzindo, por ex., instruções nop no código; alterando a ordem das instruções; ...)

Stall; Forwarding ou bypassing

Data Hazards

- **Read After Write (RAW)**

  $Instr_J$ tenta ler o operando antes da $Instr_I$ escreve-la

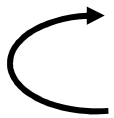


```
I: add r1,r2,r3
J: sub r4,r1,r3
```

  - **Causada por uma "Dependência" (nomenclatura de compiladores).**

Data Hazards

- **Write After Read (WAR)**
  $Instr_J$ escreve o operando ***antes*** que a $Instr_I$ o leia

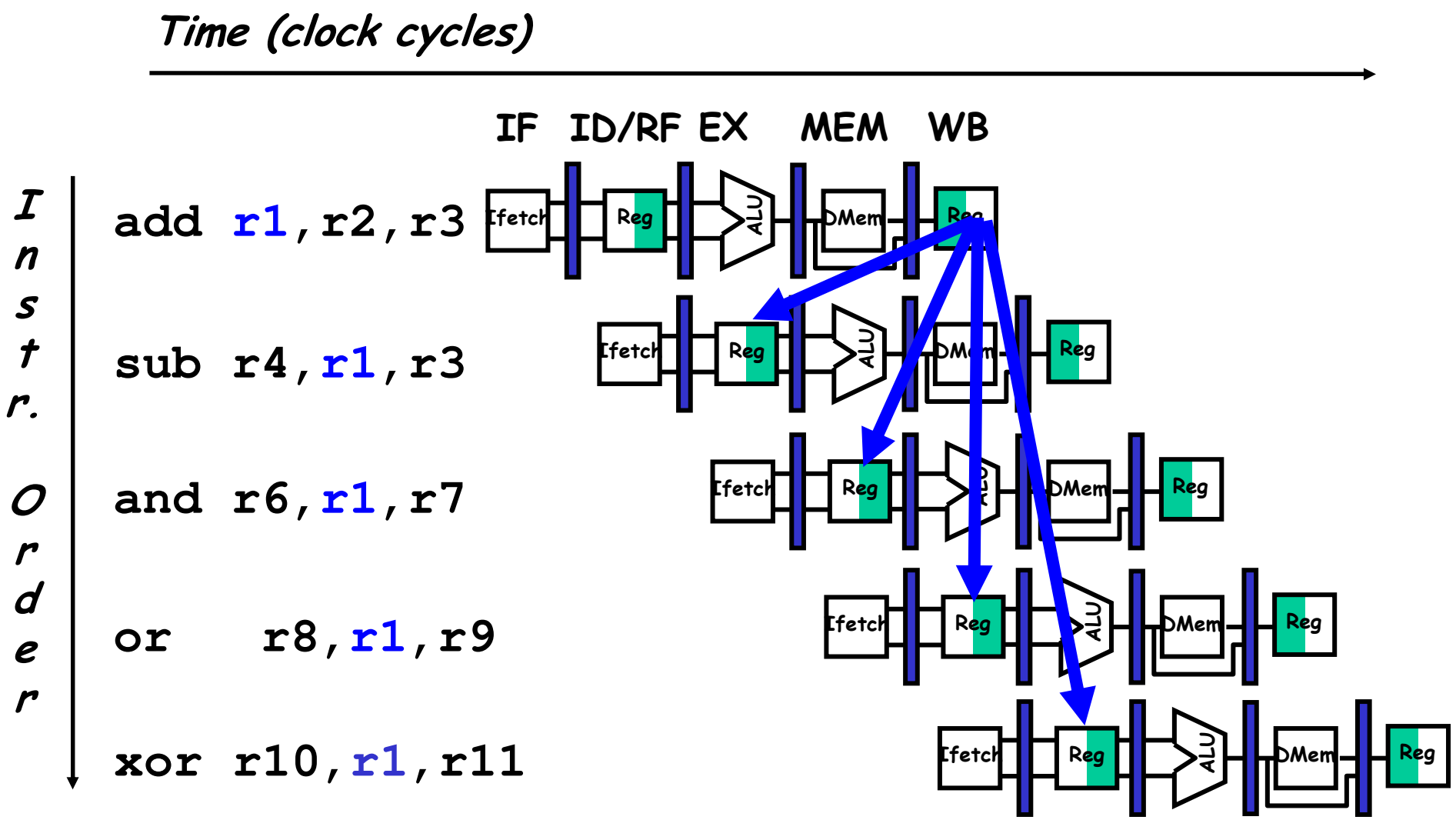


```
I: sub r4,r1,r3
J: add r1,r2,r3
K: mul r6,r1,r7
```

- Chamada "anti-dependência" (nomenclatura de compiladores). Devido ao reuso do nome "r1".

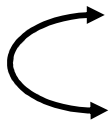
- Não ocorre no pipeline do MIPS:
  - Todas instruções usam 5 estágios, e
  - Leituras são no estágio 2, e
  - Escritas são no estágio 5

Data Hazard em R1



Data Hazards

- **Write After Write (WAW)**
  $Instr_J$ escreve o operando *antes* que a $Instr_I$ o escreva.



```
I: sub r1,r4,r3
J: add r1,r2,r3
K: mul r6,r1,r7
```

- **Chamada "dependência de saída" (nomenclatura de compiladores). Devido ao reuso do nome "r1".**

- **Não ocorre no pipeline do MIPS:**
  - **Todas Instruções são de 5 estágios, e**
  - **Escritas são sempre no 5 estágio**
  - **(WAR e WAW ocorrem em pipes mais sofisticados)**

Data Hazards – Solução SW

- Compilador reconhece o data hazard e adiciona nops
- Compilador reconhece o data hazard e troca a ordem das instruções (quando possível)

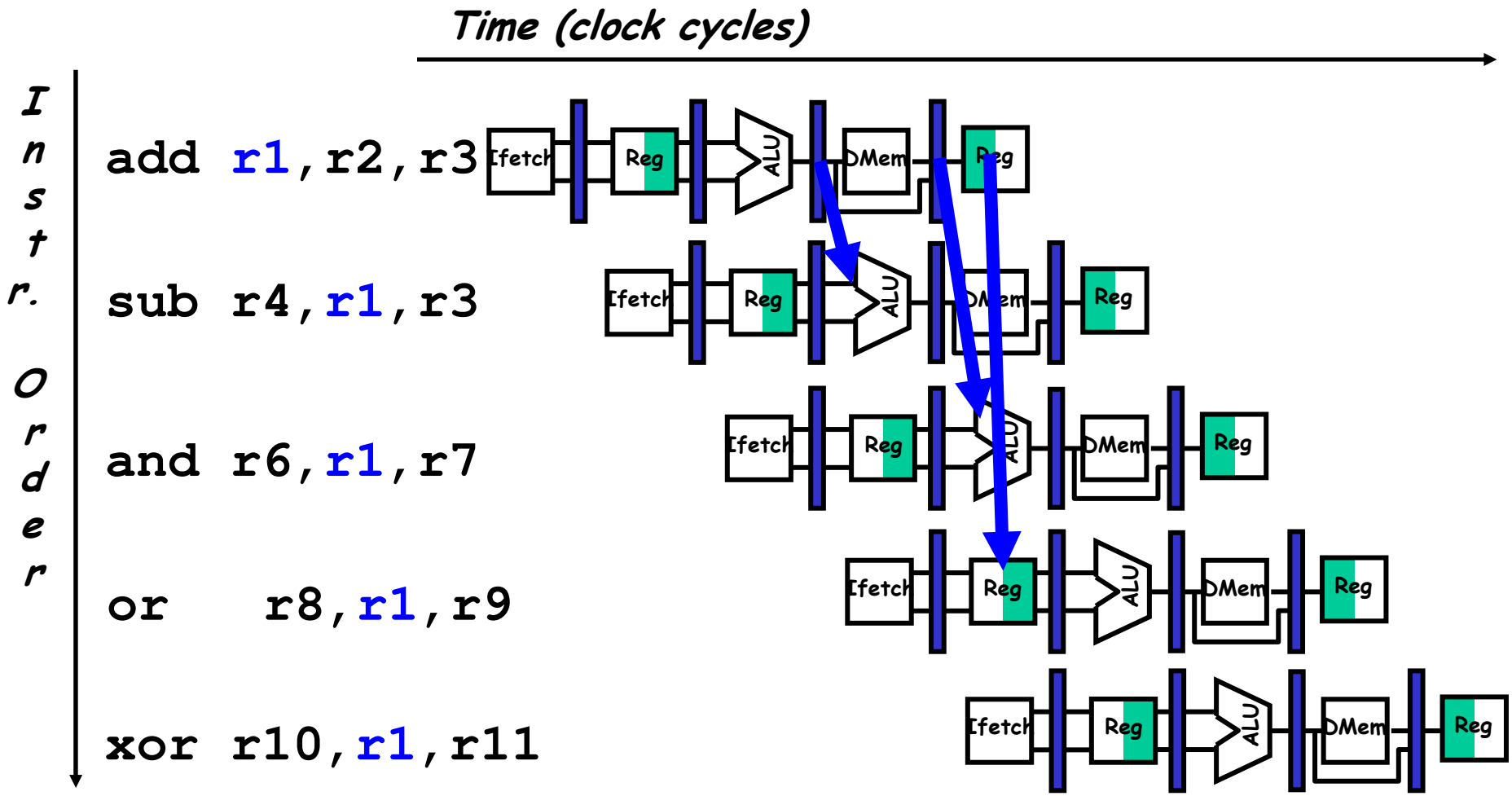
Exemplo:

```
sub  R2, R1, R3     ; reg R2 escrito por sub
nop                 ; no operation
nop
nop
and R12, R2, R5     ; resultado do sub disponível
or   R13, R6, R2
add R14, R2, R2
sw   100 (R2), R15
```

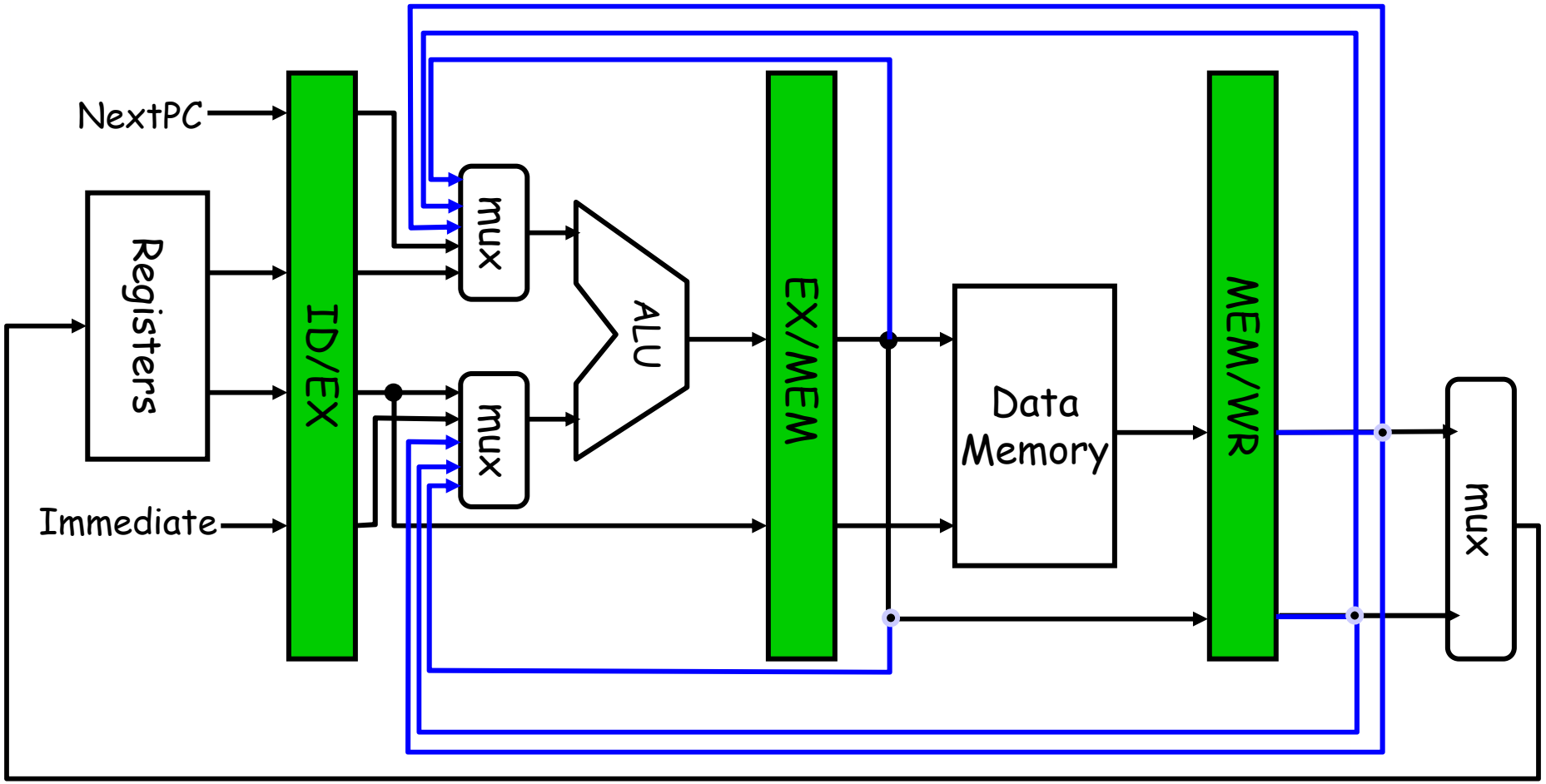
Data Hazard Control: Stalls

- Hazard ocorre quando a instr. Lê (no estágio ID) um reg que será escrito, por uma instr. anterior (no estágio ID)
- Solução: Detectar o hazard e parar a instrução no pipeline até o hazard ser resolvido
- Detectar o hazard pela comparação do campo read no IF/ID pipeline register com o campo write dos outros pipeline registers (ID/EX, EX/MEM, MEM/WB)
- Adicionar bubble no pipeline
  - Preservar o PC e o IF/ID pipeline register

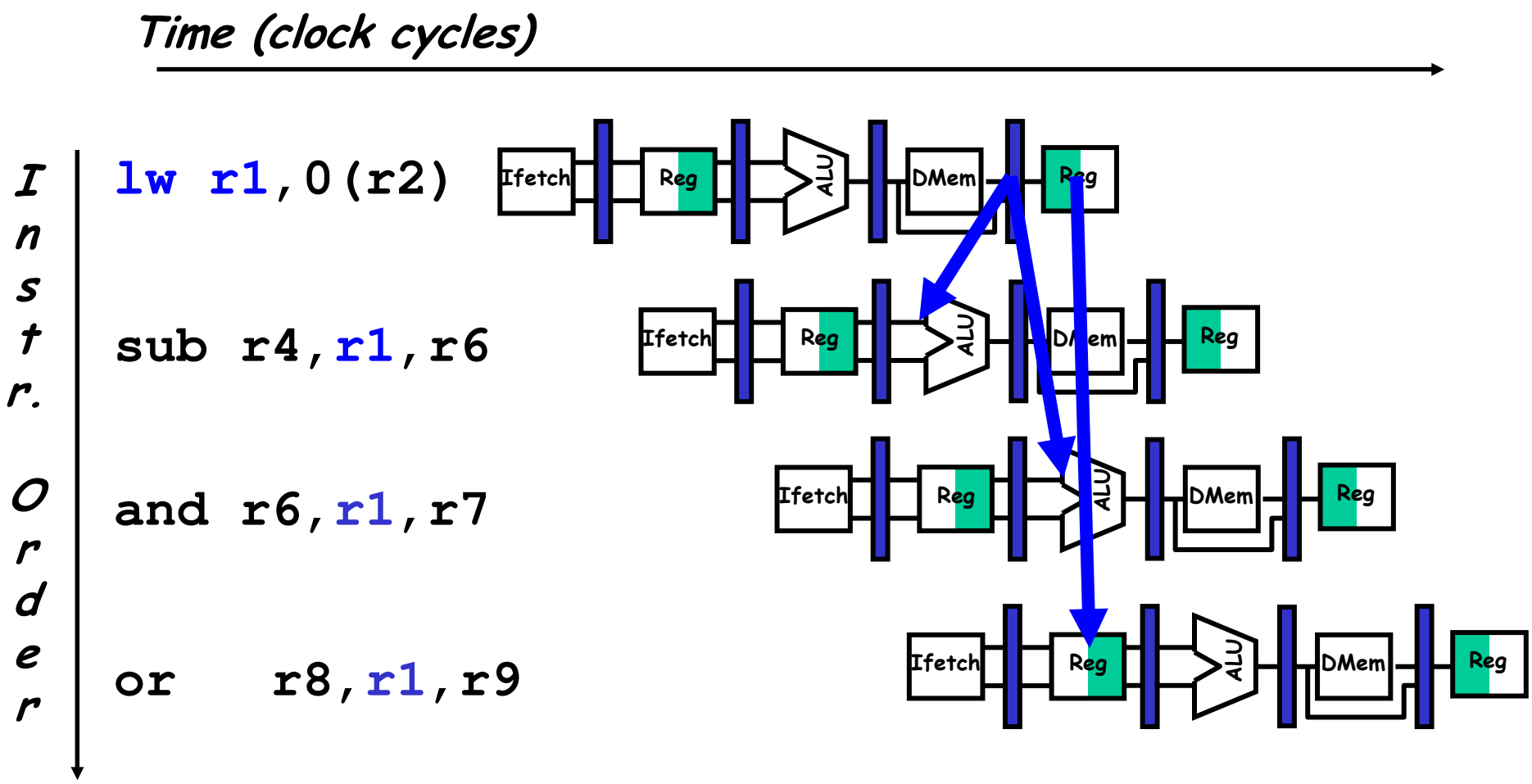
Redução do Data Hazard - Forwarding



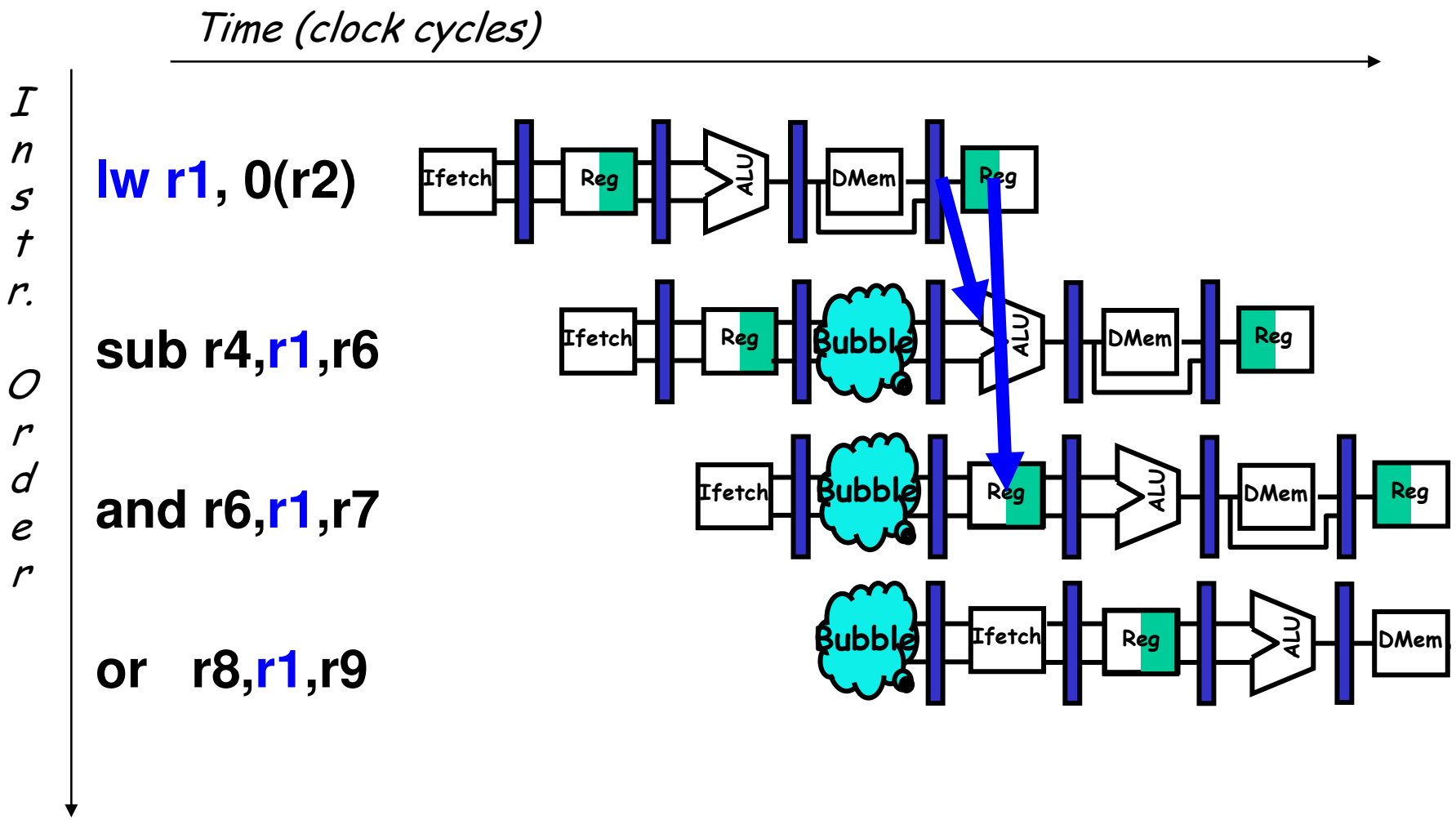
HW para Forwarding



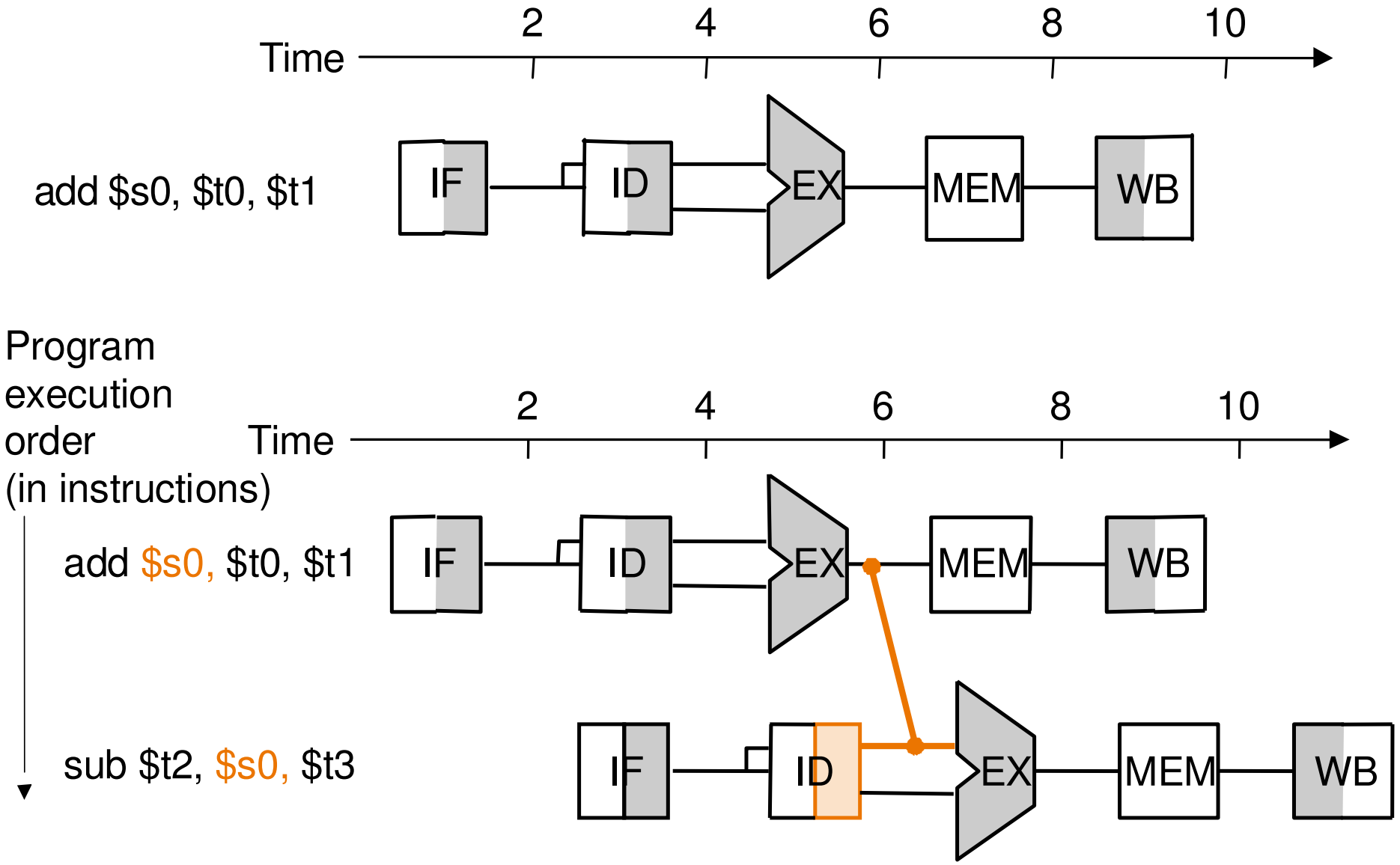
Data Hazard com Forwarding



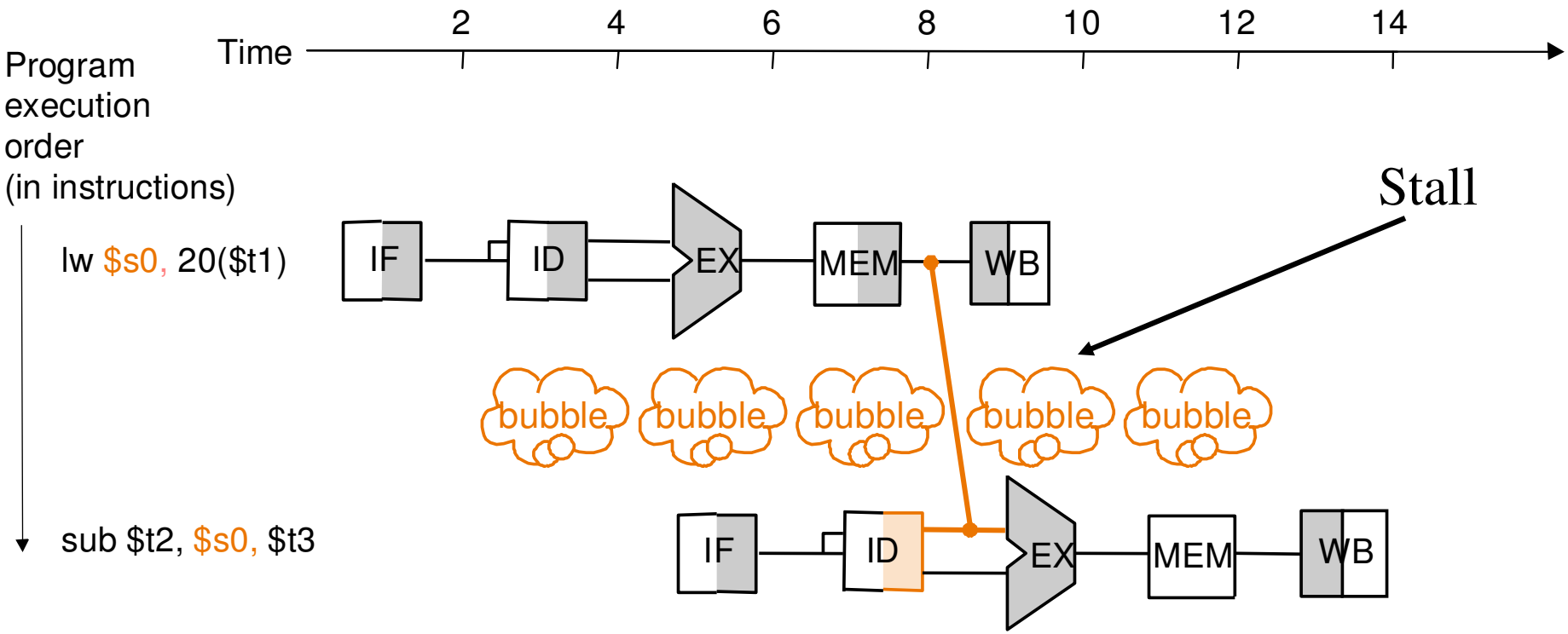
Data Hazard com Forwarding



Hazard de Dados



Hazard de Dados



Exemplo

- Encontre o hazard no código abaixo e resolva-o:

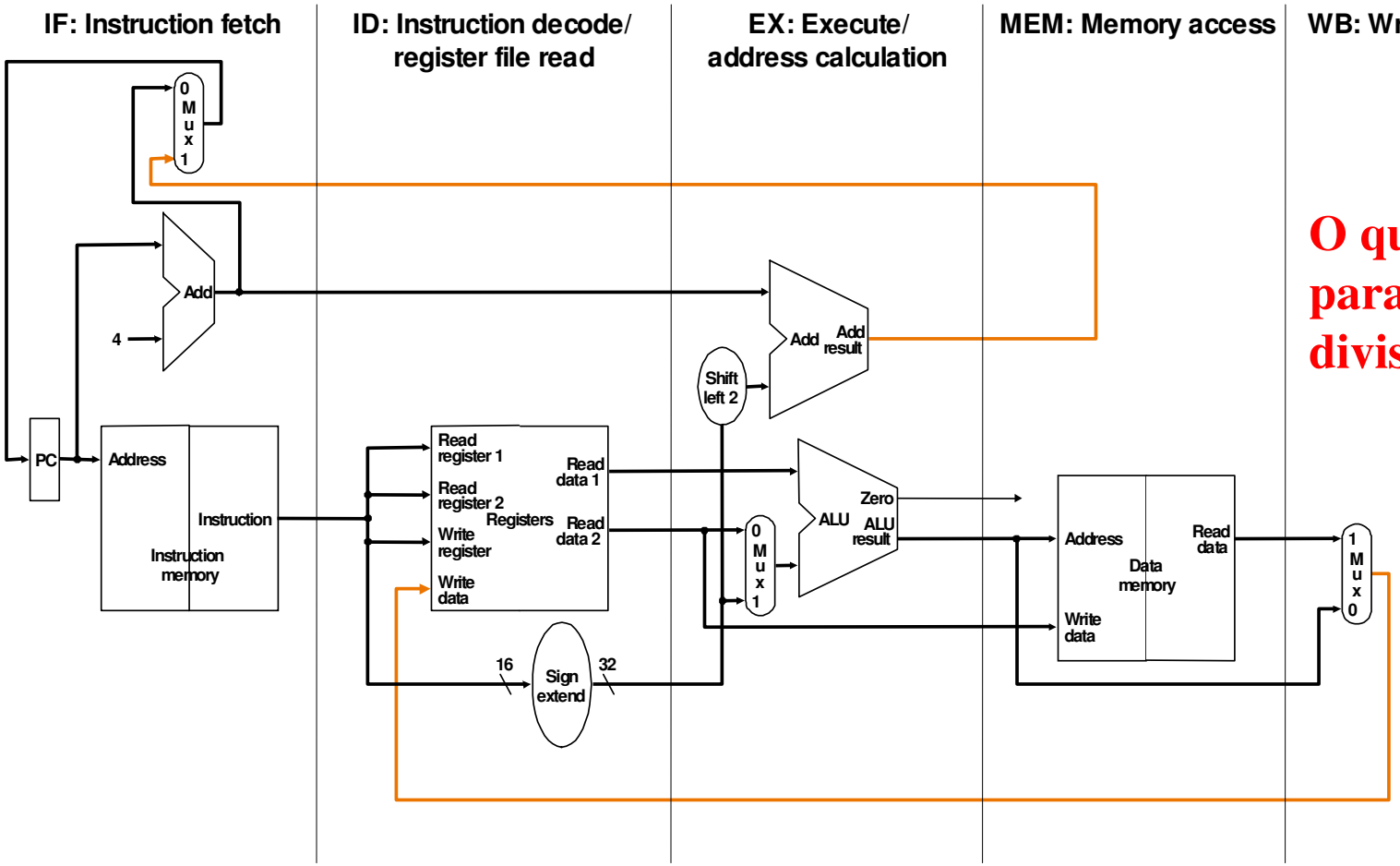
```
                                  # $t1 tem o end. de v[k]
lw      $t0, 0($t1)               # $t0 = v[k]
lw      $t2,4($t1)                # $t2 = v[k+1]
sw      $t2, 0($t1)               # v[k] = $t2
sw      $t0, 4($t1)               # v[k+1] = $t0
```

Solução:

```
                                  # $t1 tem o end. de v[k]
lw      $t0, 0($t1)               # $t0 = v[k]
lw      $t2,4($t1)                # $t2 = v[k+1]
sw      $t0, 4($t1)               # v[k+1] = $t0
sw      $t2, 0($t1)               # v[k] = $t2
```

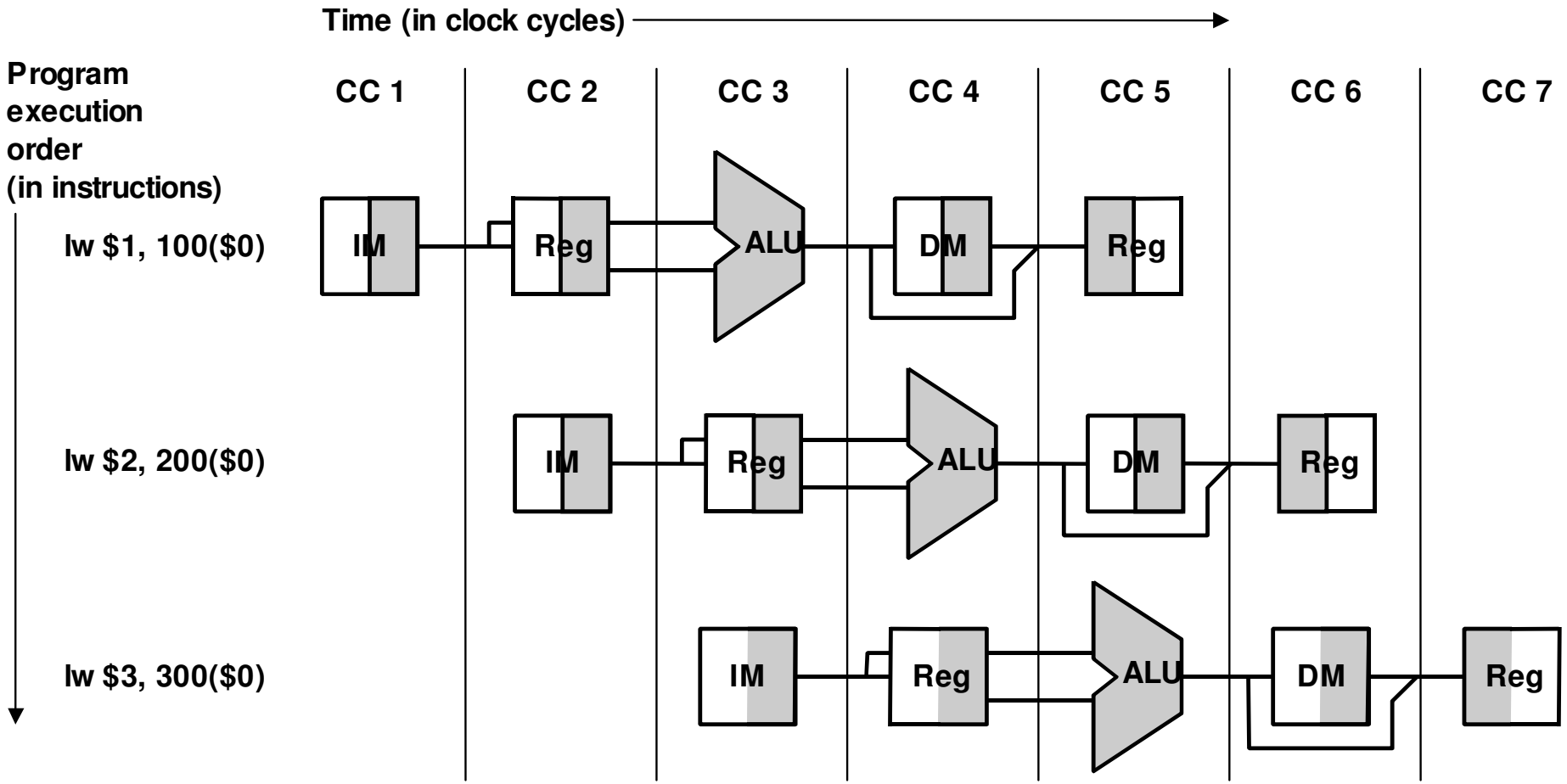
Pipeline: Idéia Básica

- 5 estágios: Fetch; Decodificação e leitura dos regs; execução ou cálculo de end. ; acesso à memória; escrita no reg. destino

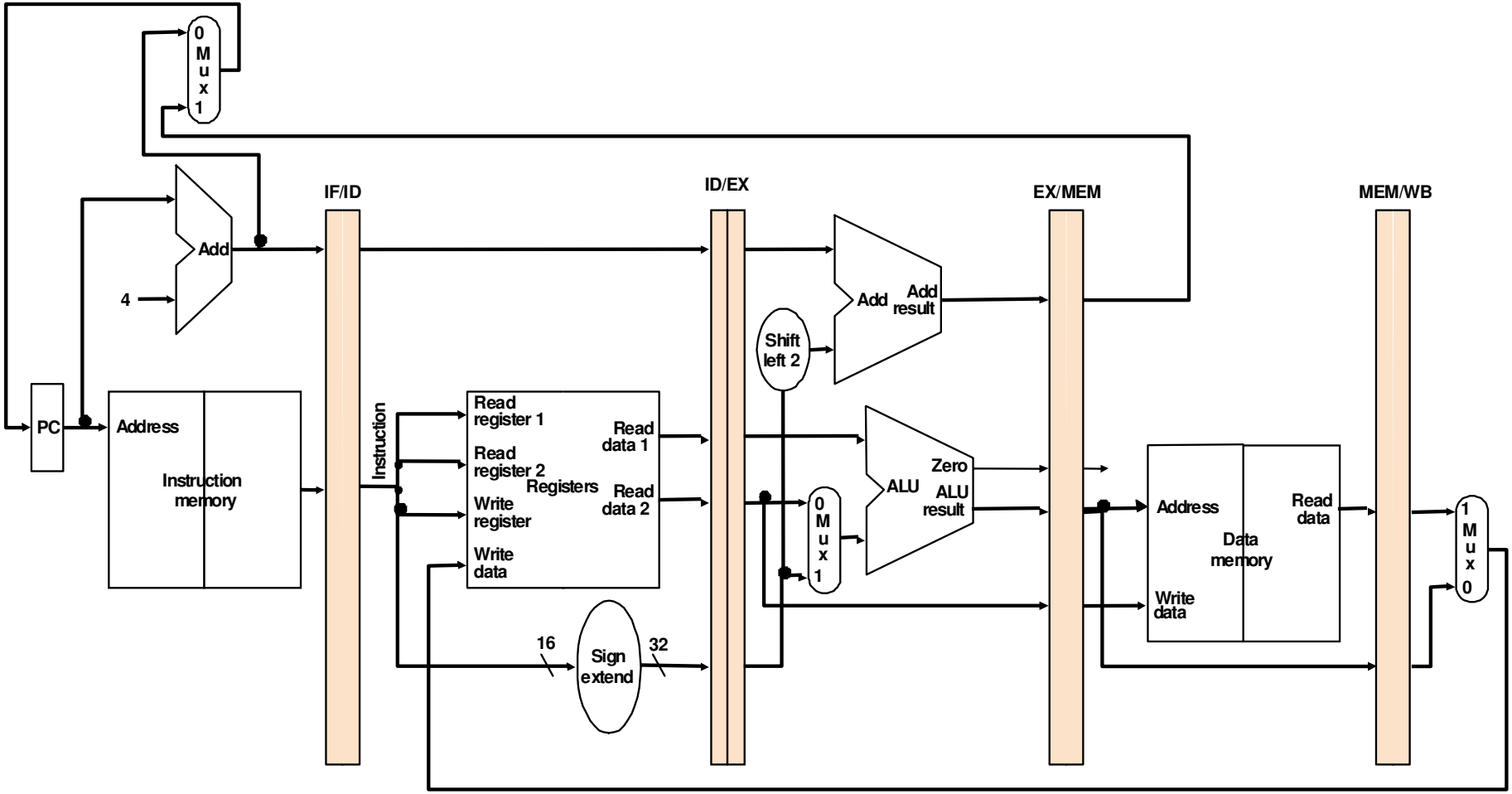


O que é necessário para tornar cada divisão em estágios?

Instruções Sendo Executadas pelo Datapath



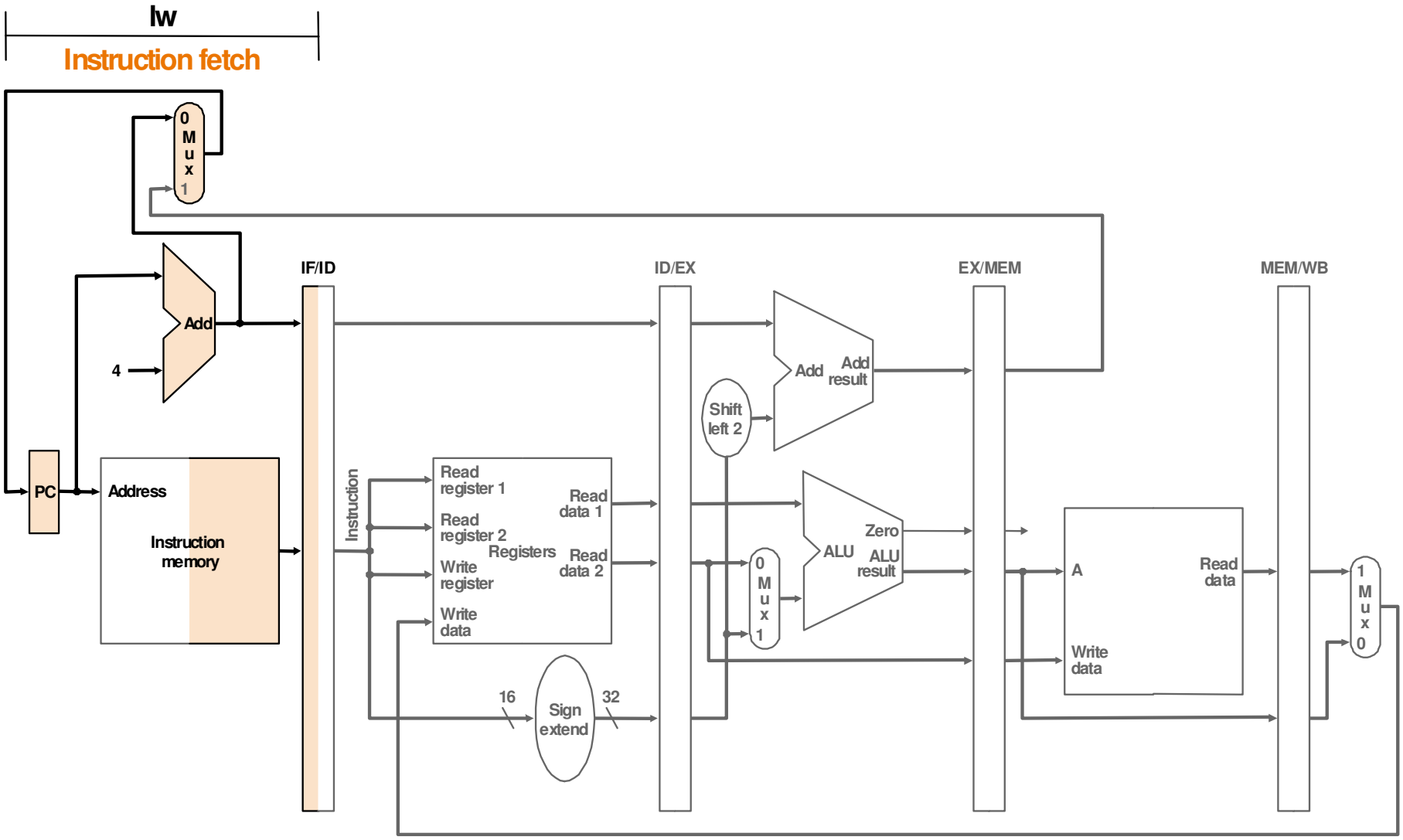
Pipelined Datapath



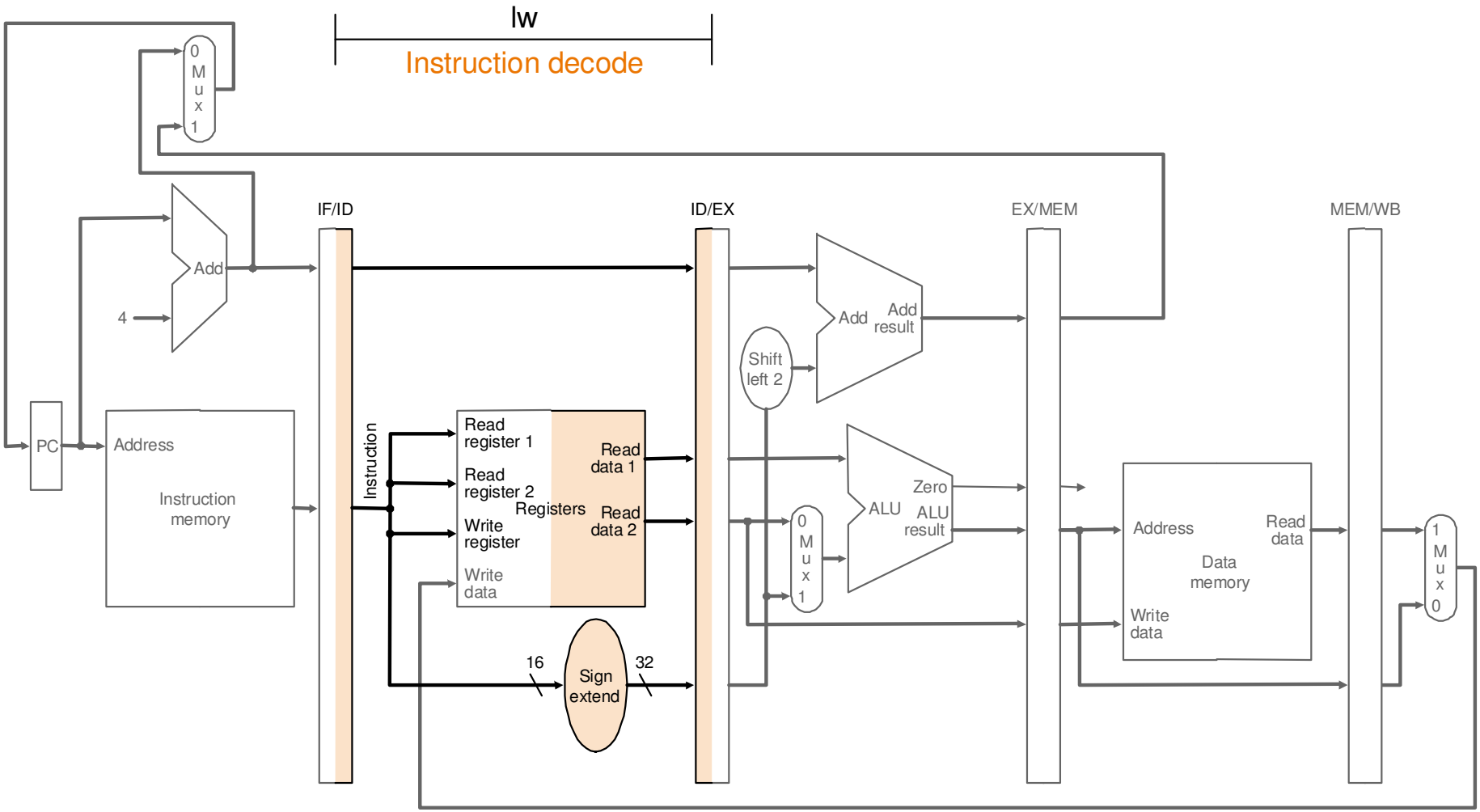
Pode acontecer algum problema nesta solução se não existir dependência de dados?

A execução de qual instrução causa o problema?

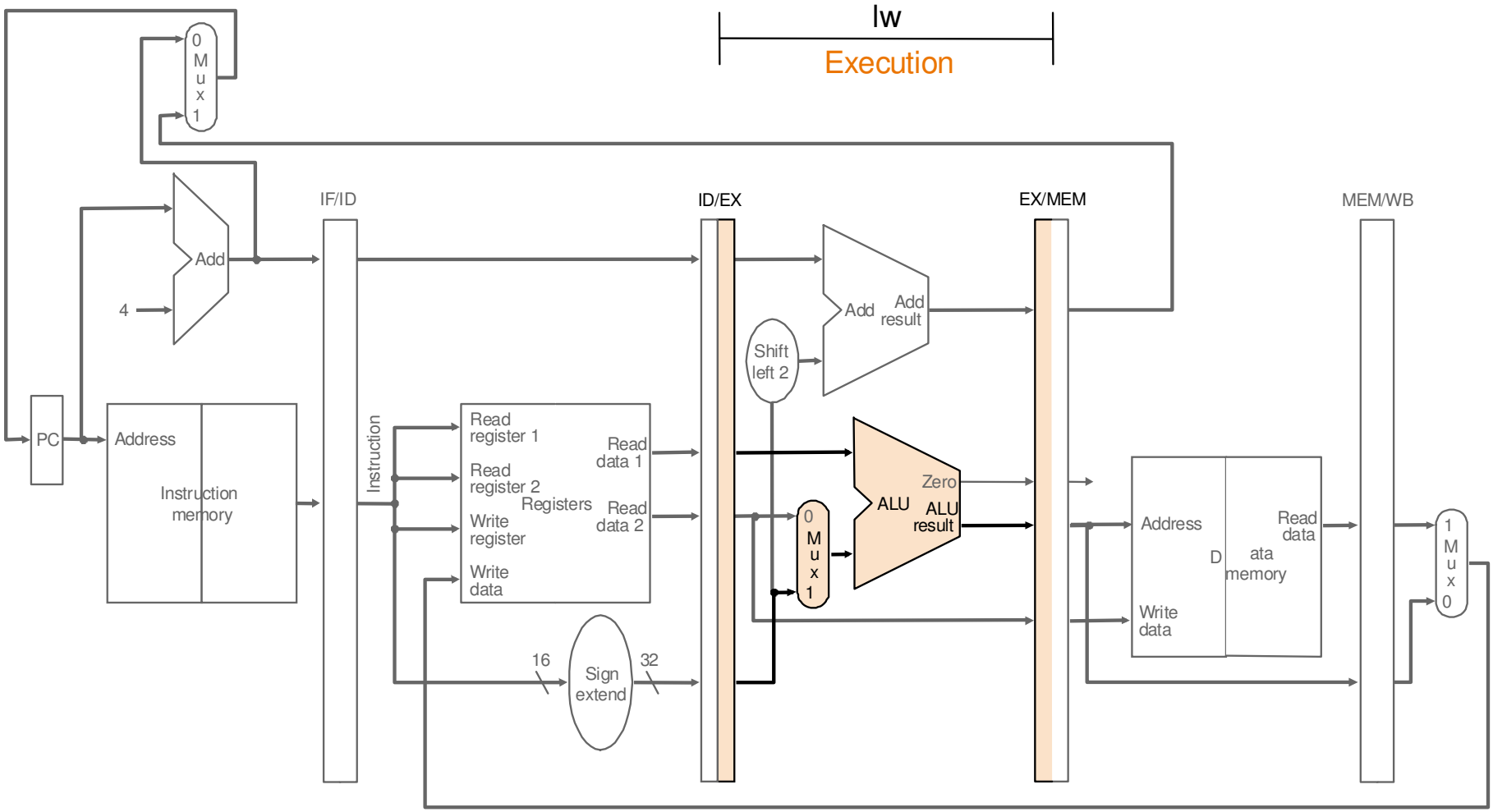
Pipelined Datapath



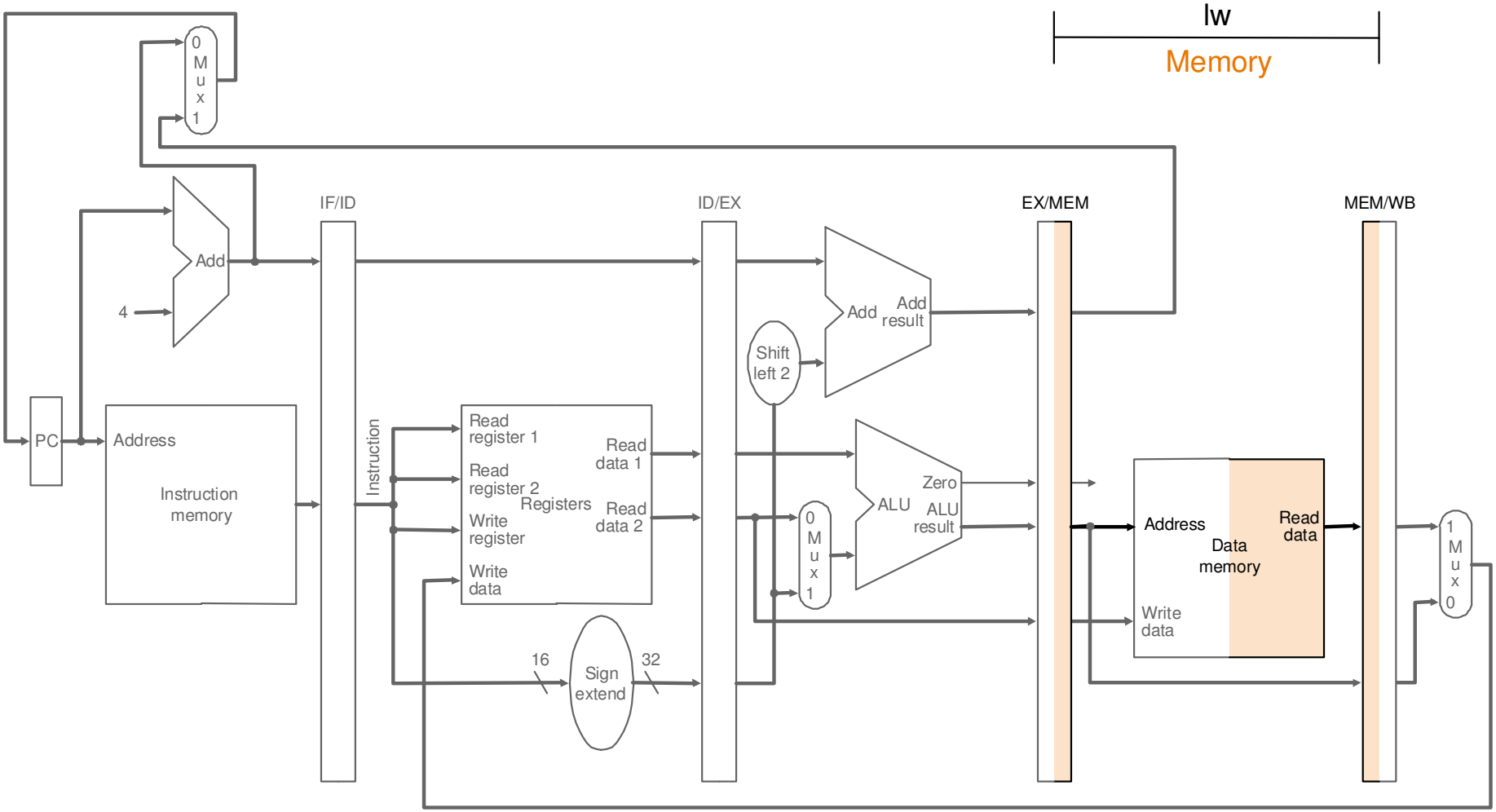
Pipelined Datapath



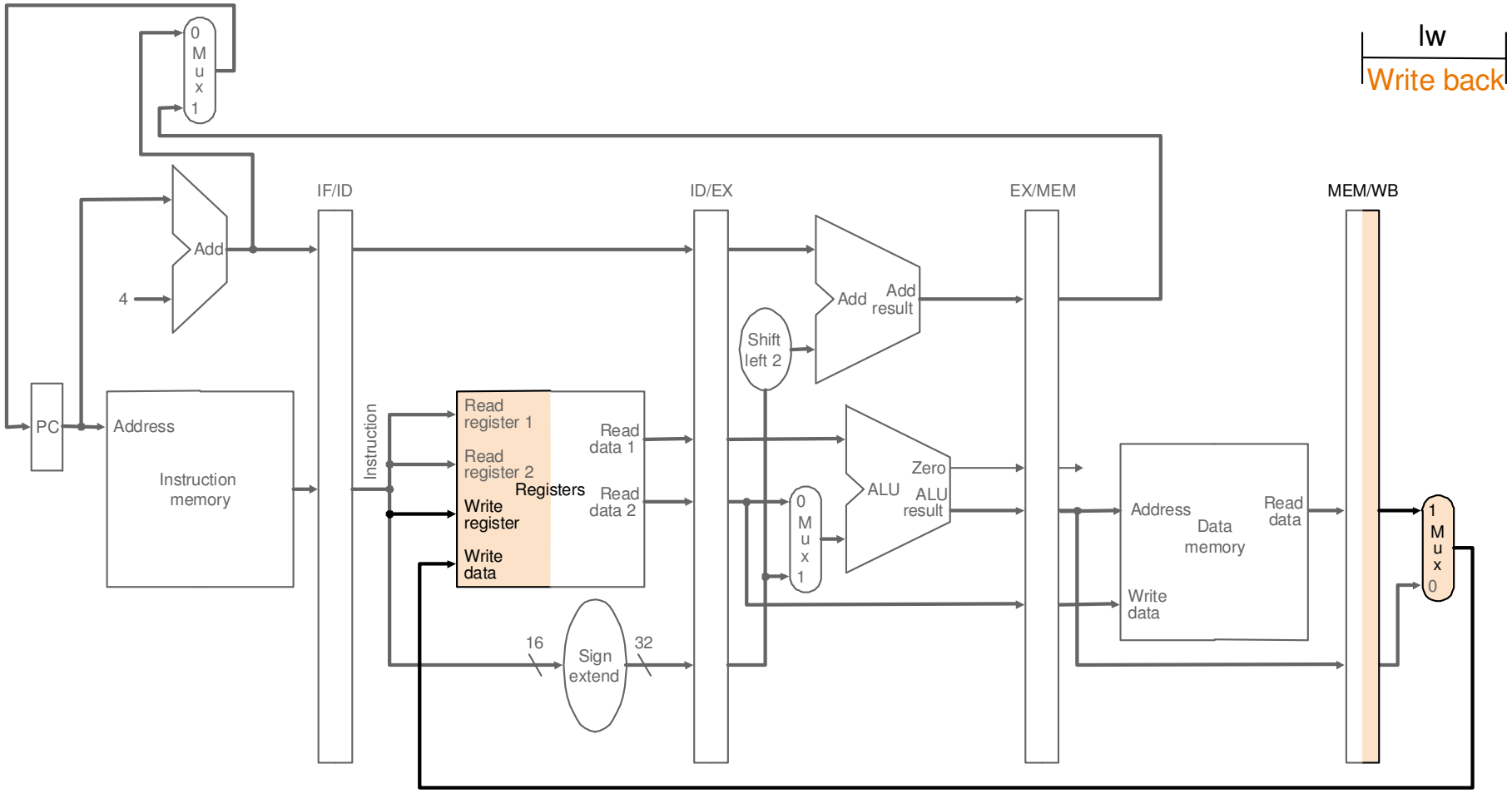
Pipelined Datapath



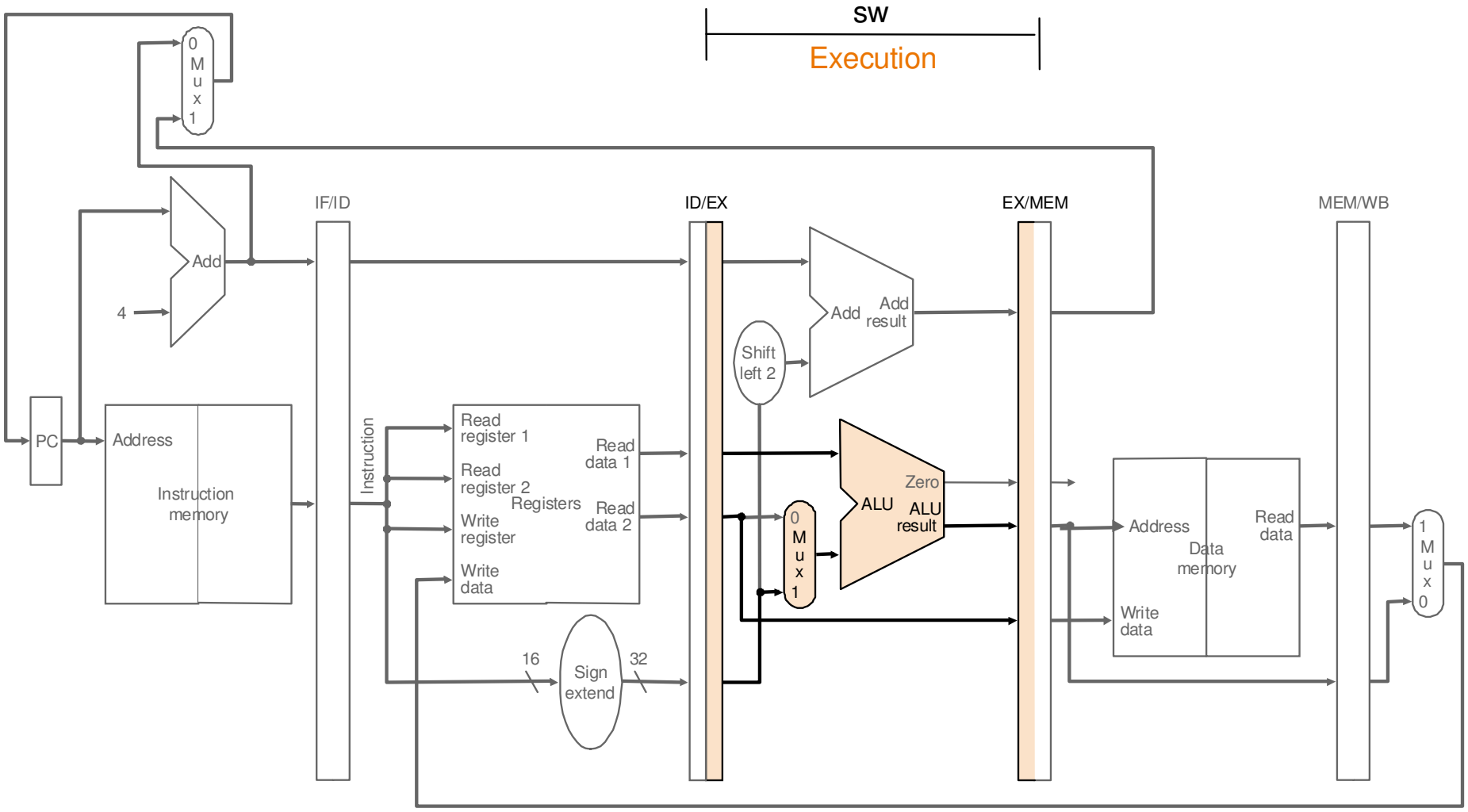
Pipelined Datapath



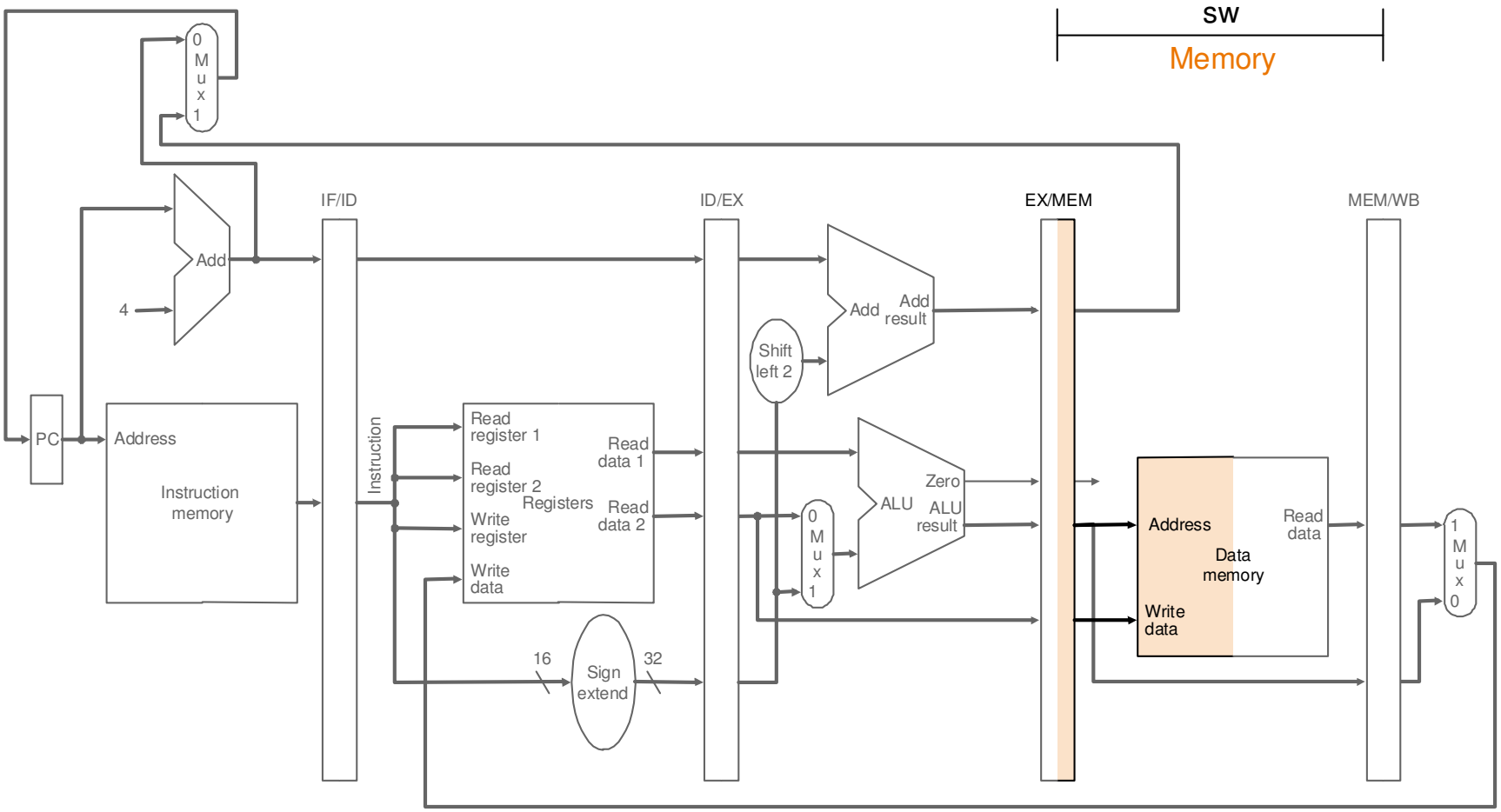
Pipelined Datapath



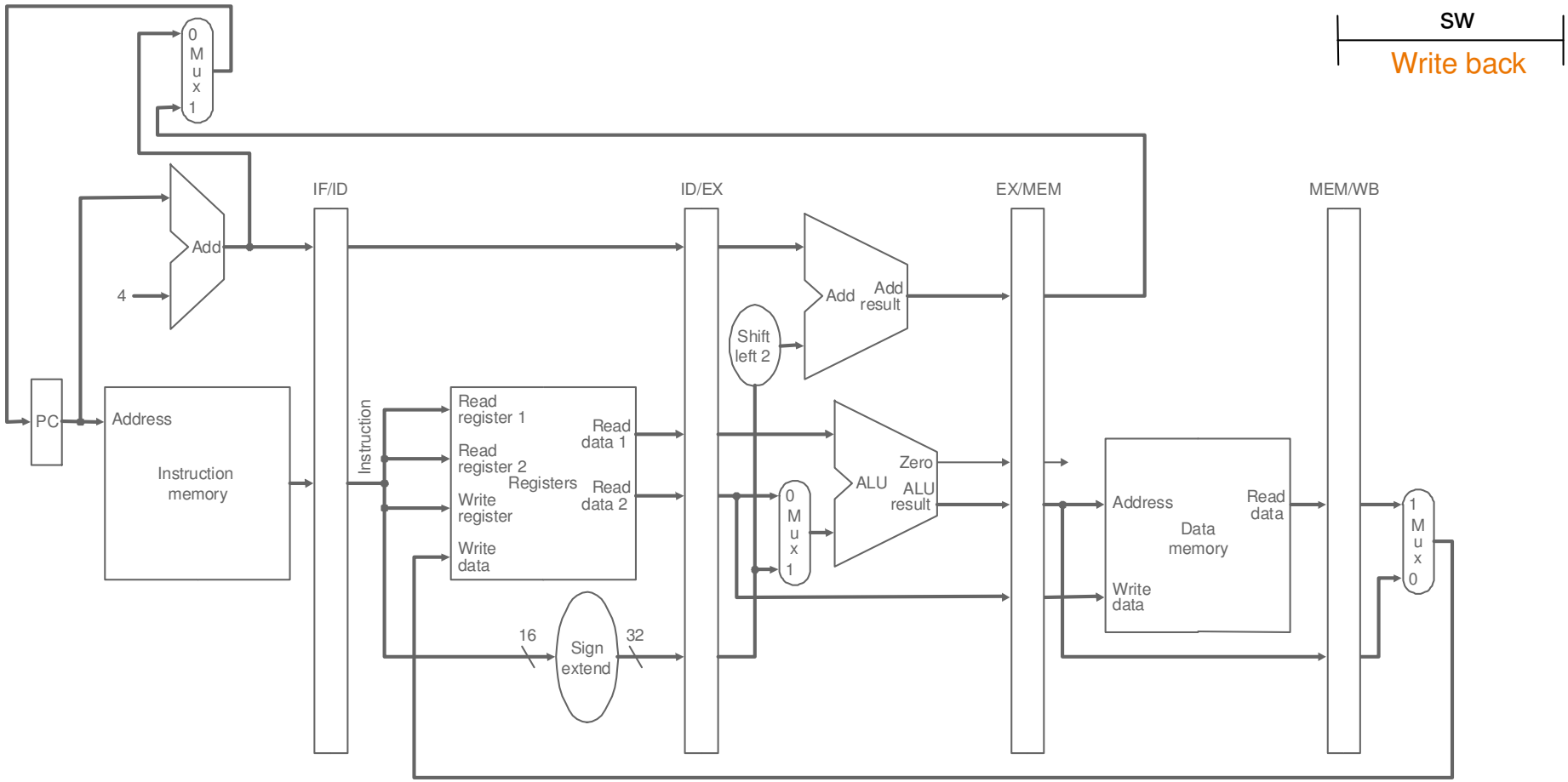
Pipelined Datapath



Pipelined Datapath

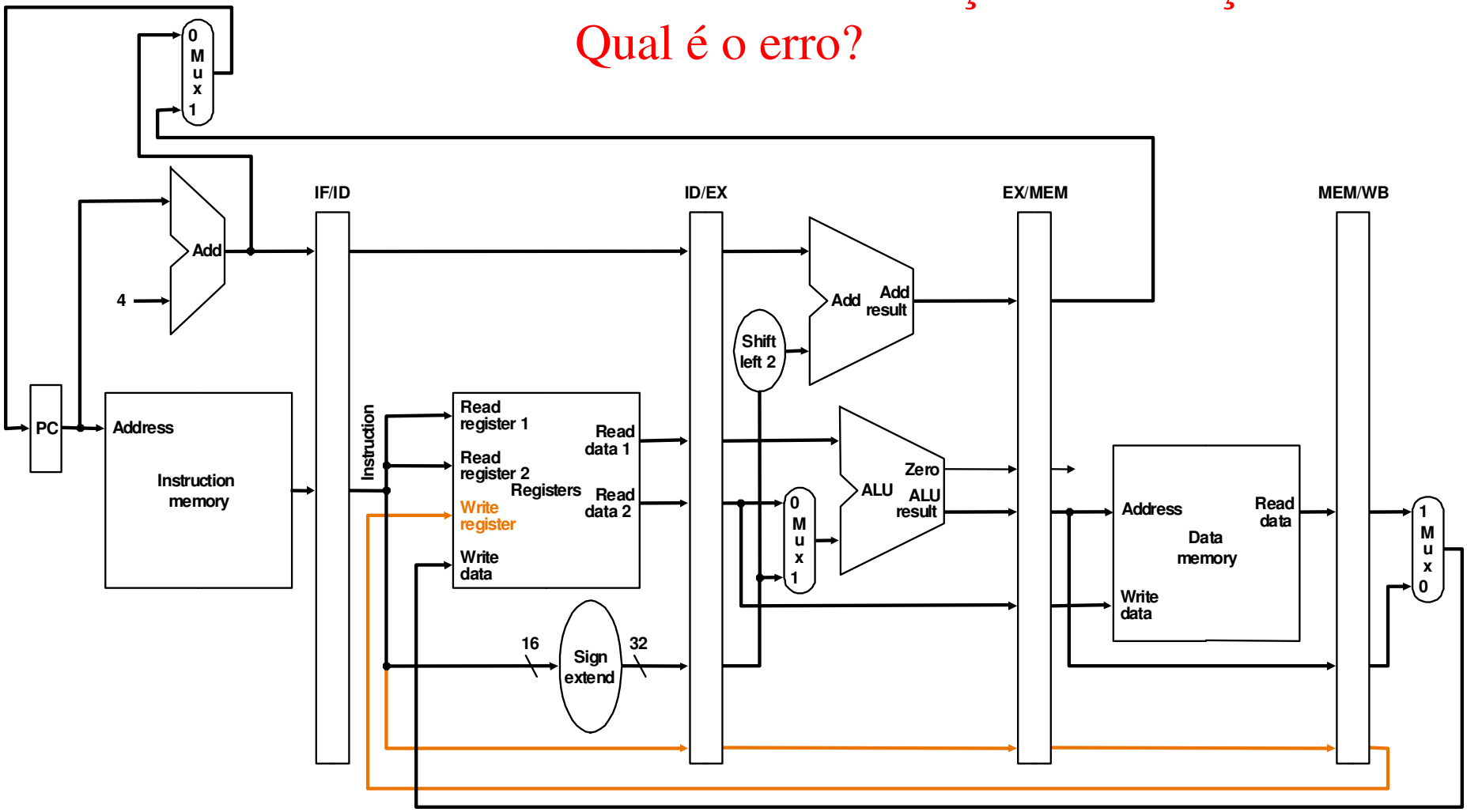


Pipelined Datapath

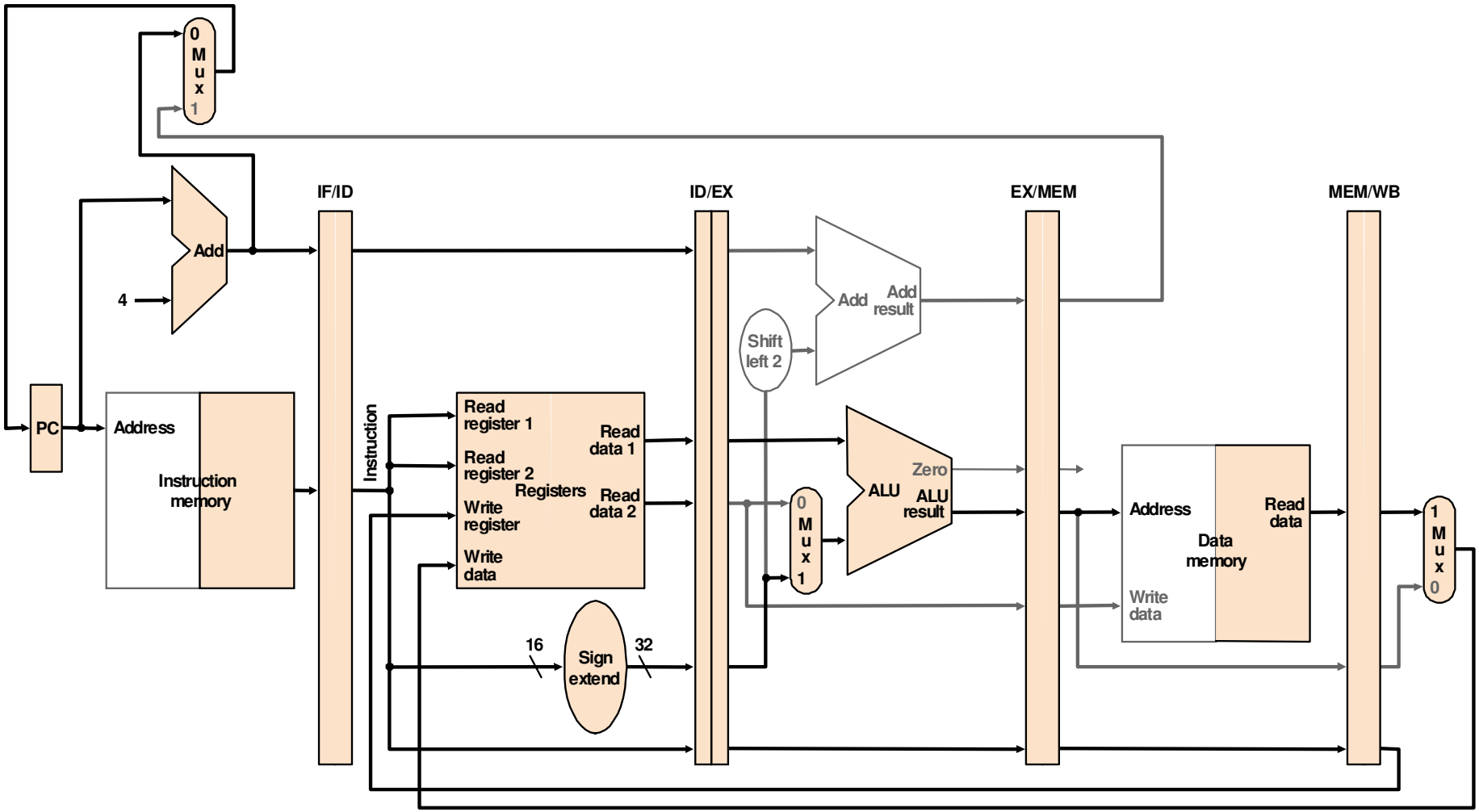


Datapath Correto

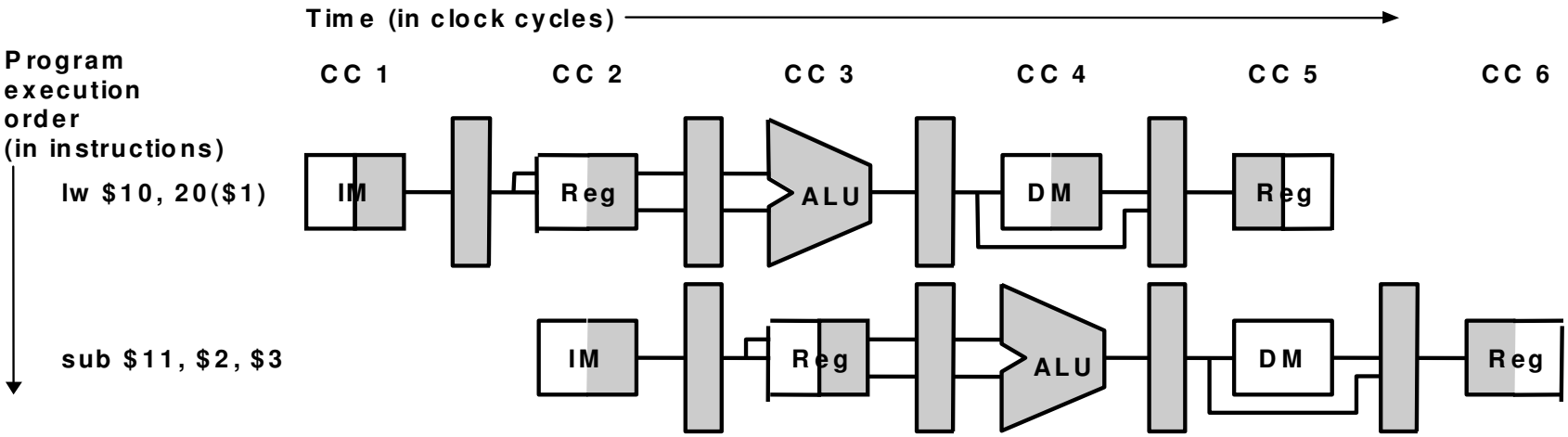
Problema na execução da instrução load.
Qual é o erro?



Datapath com os estágios usados para um instrução lw



Representações Gráfica do Pipeline



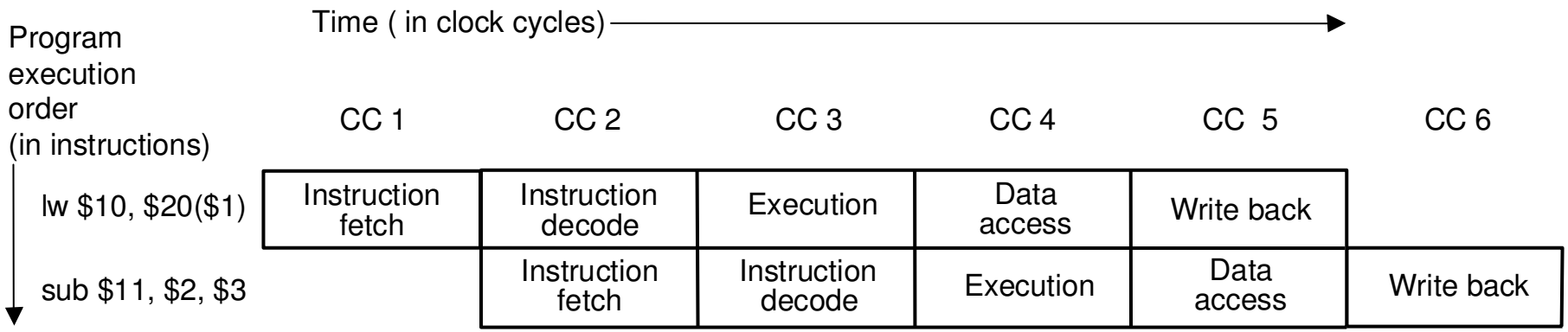
Ajuda a responder perguntas como:

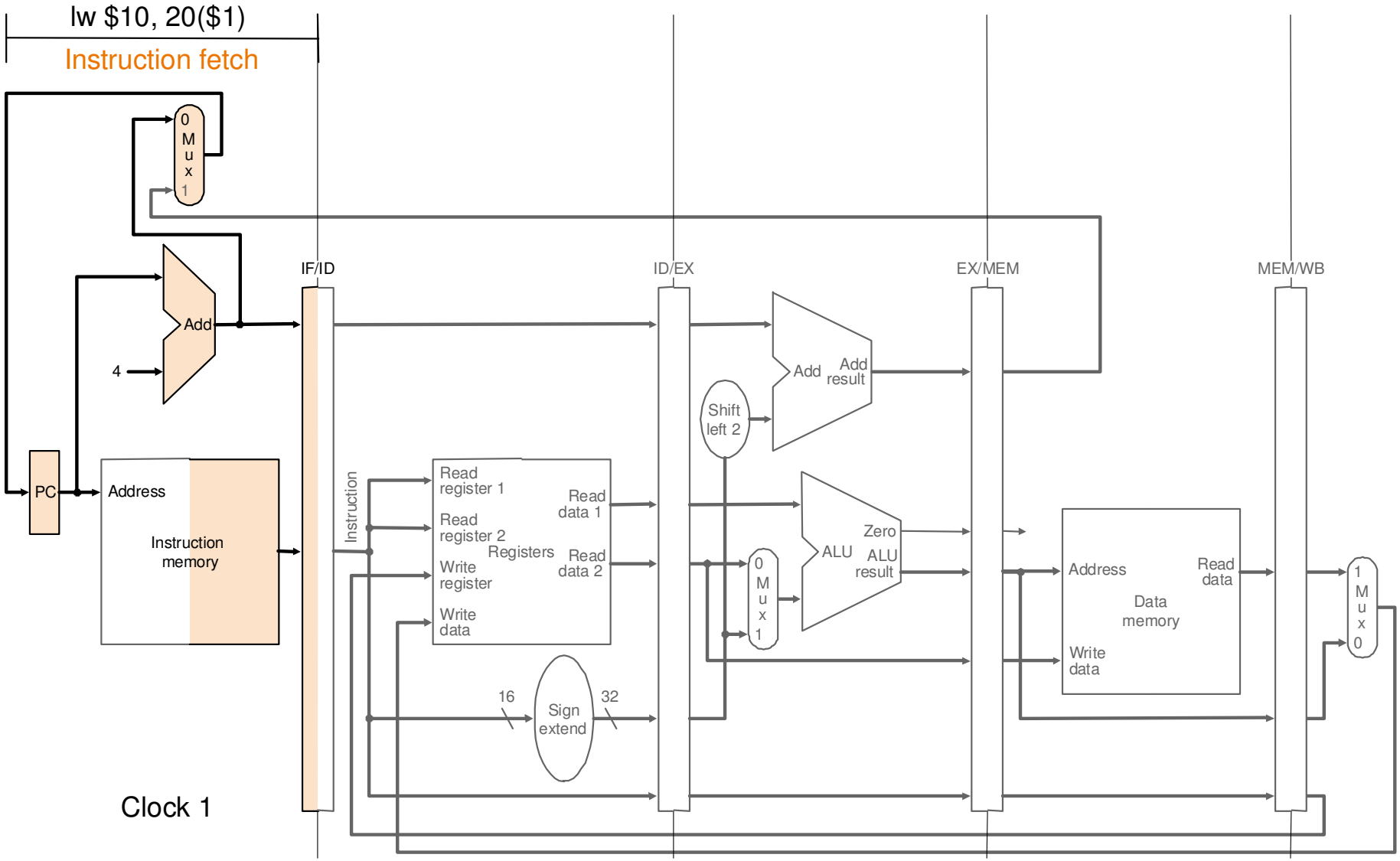
Quantos ciclos são gasto para executar este código?

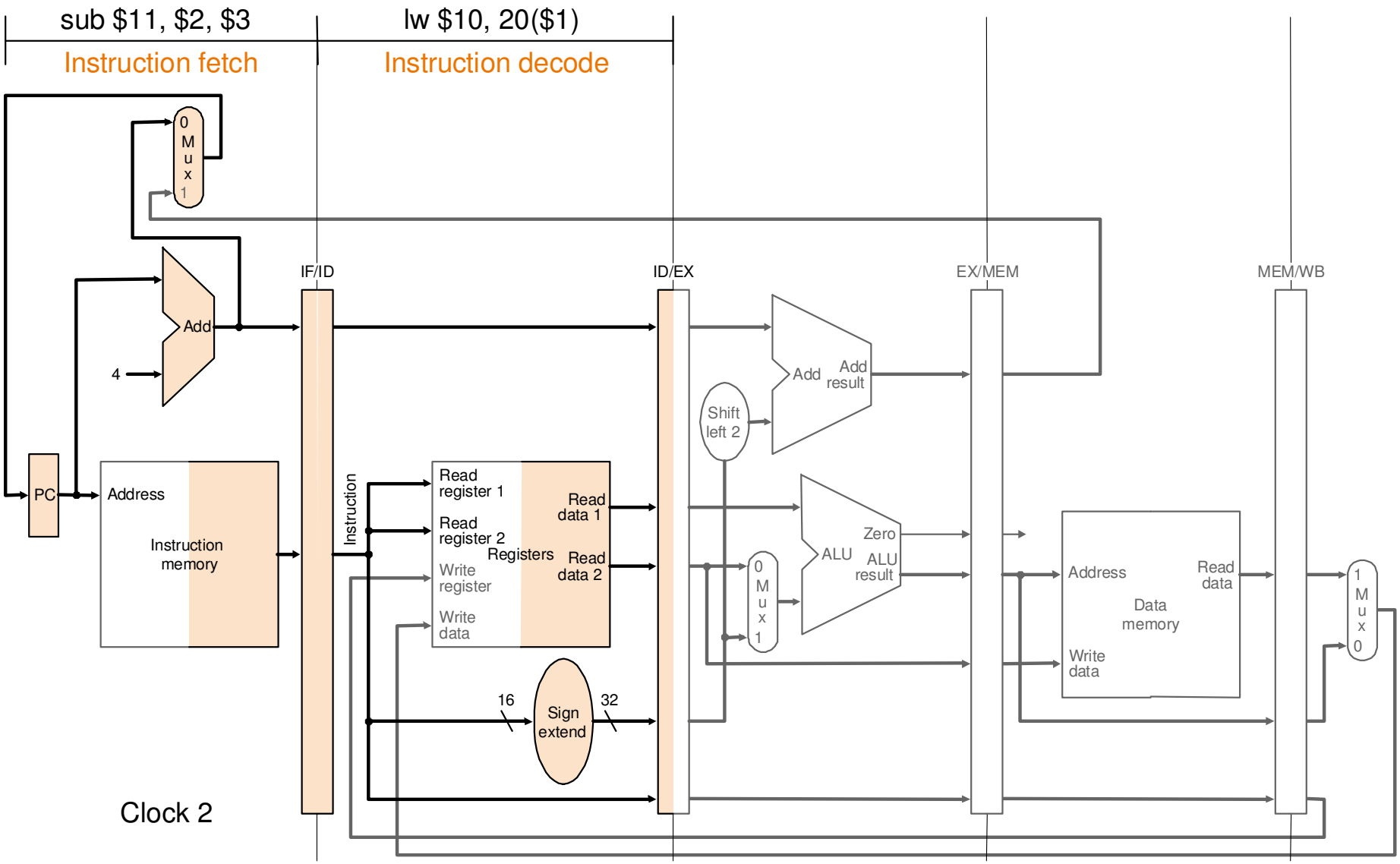
O que a ALU está fazendo durante o ciclo 10?

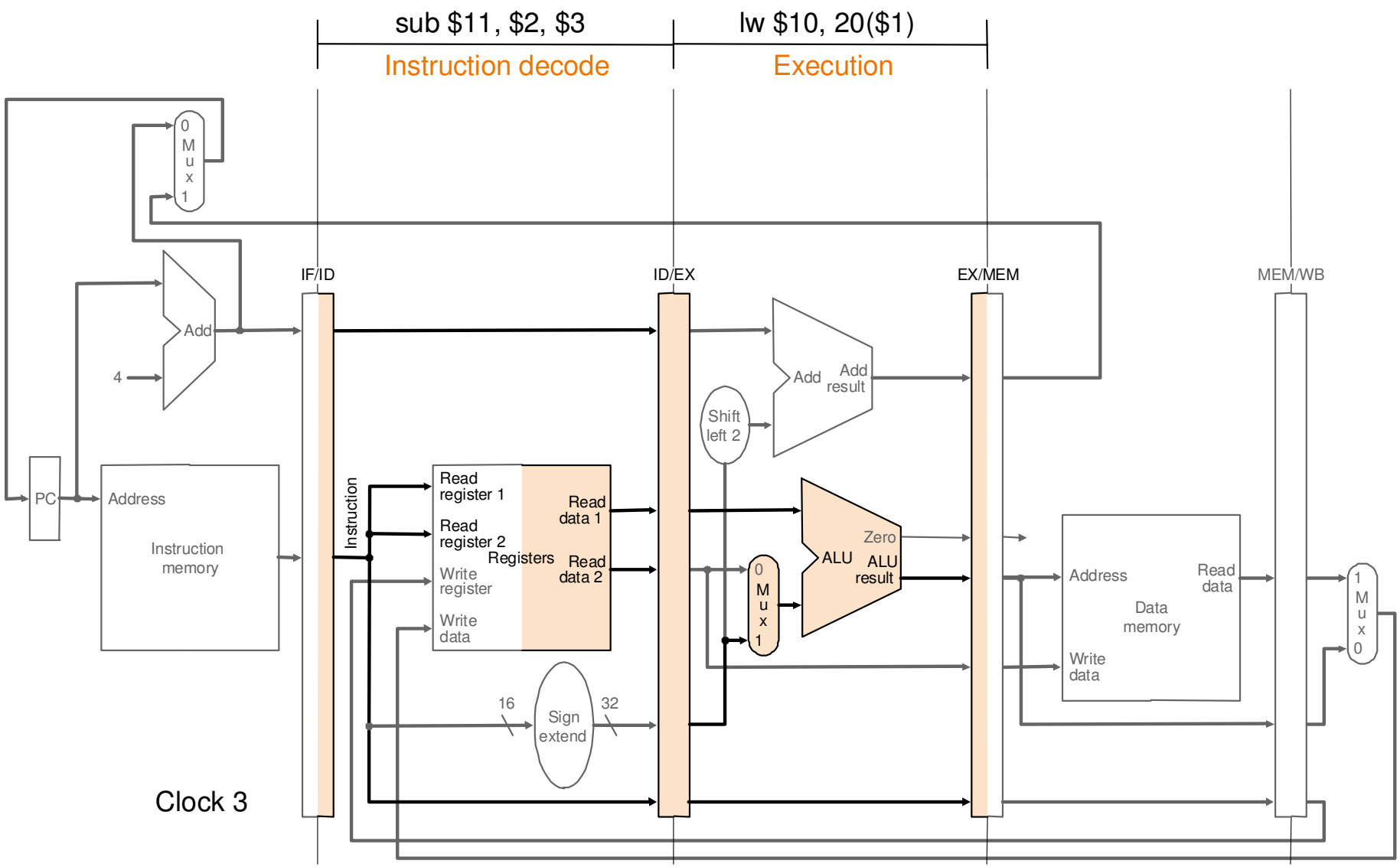
Ajuda a entender os datapaths

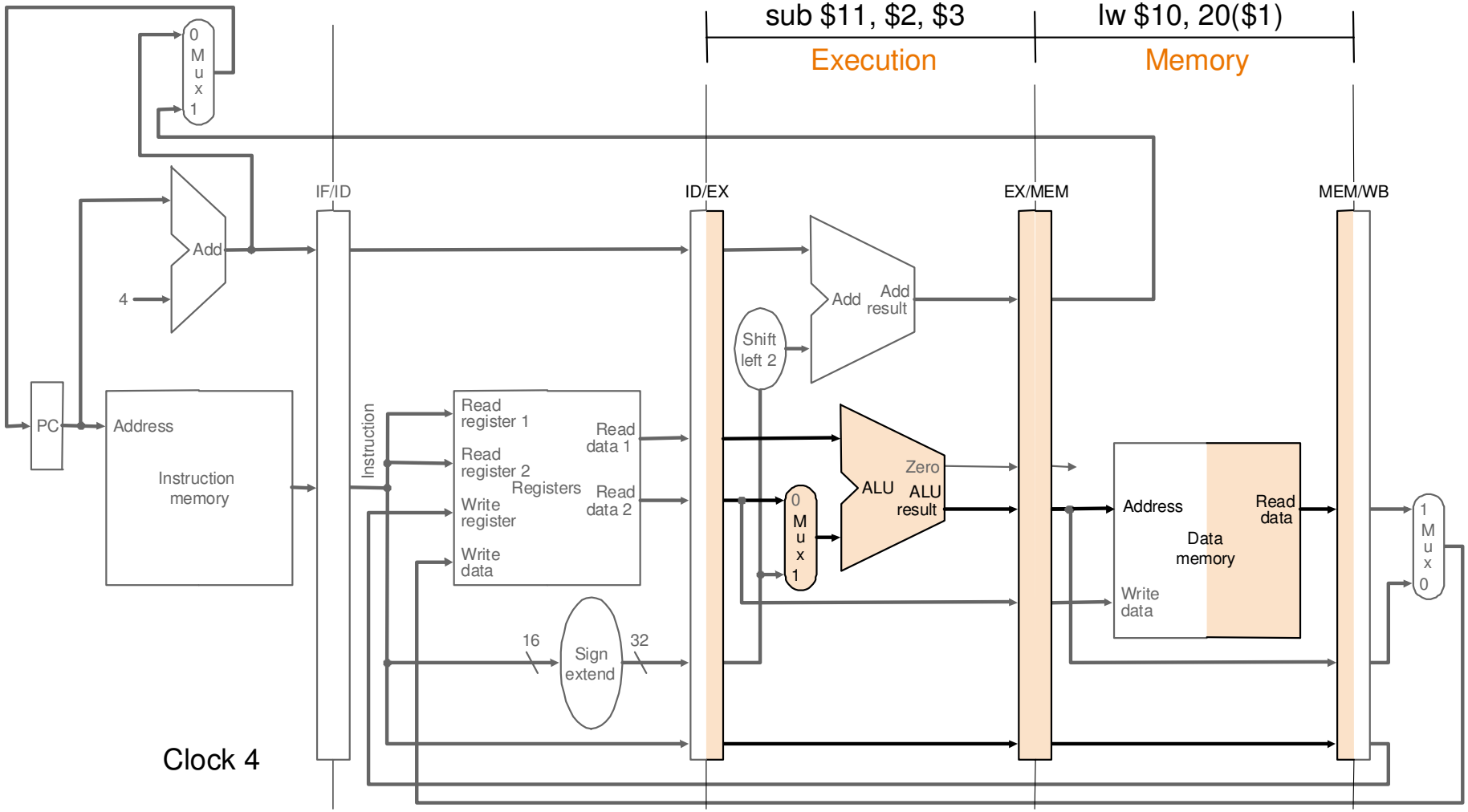
Representações Gráfica do Pipeline

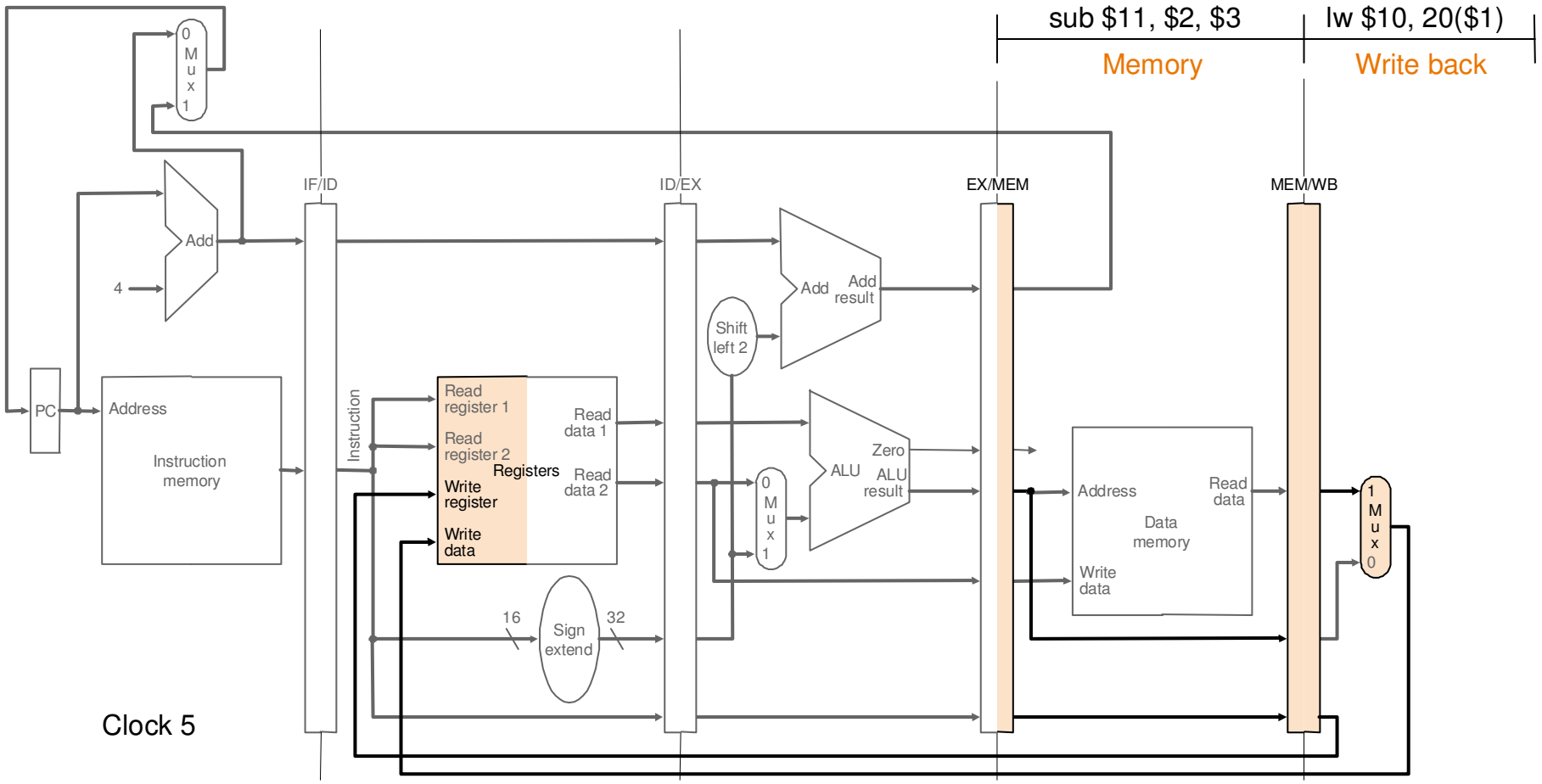


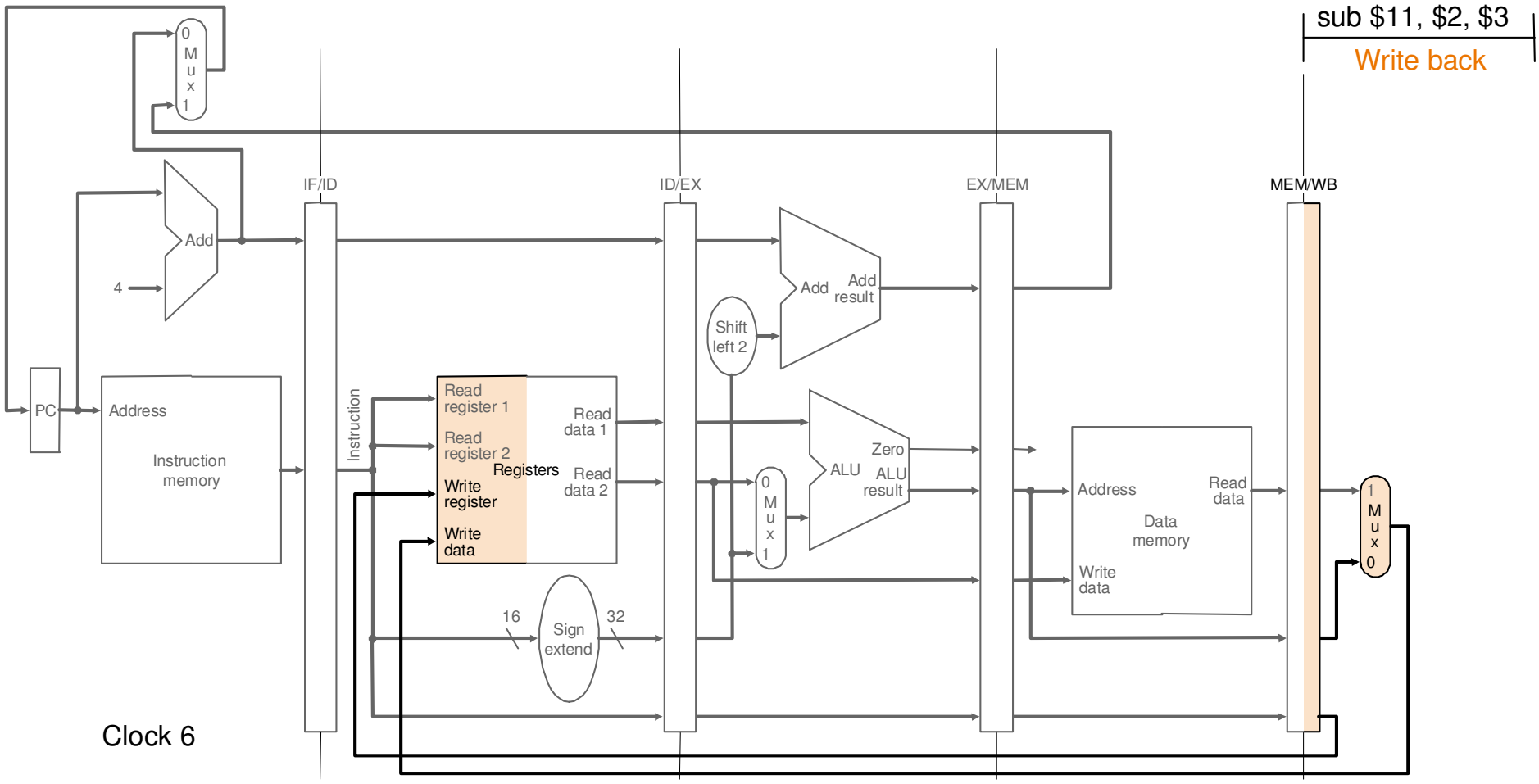




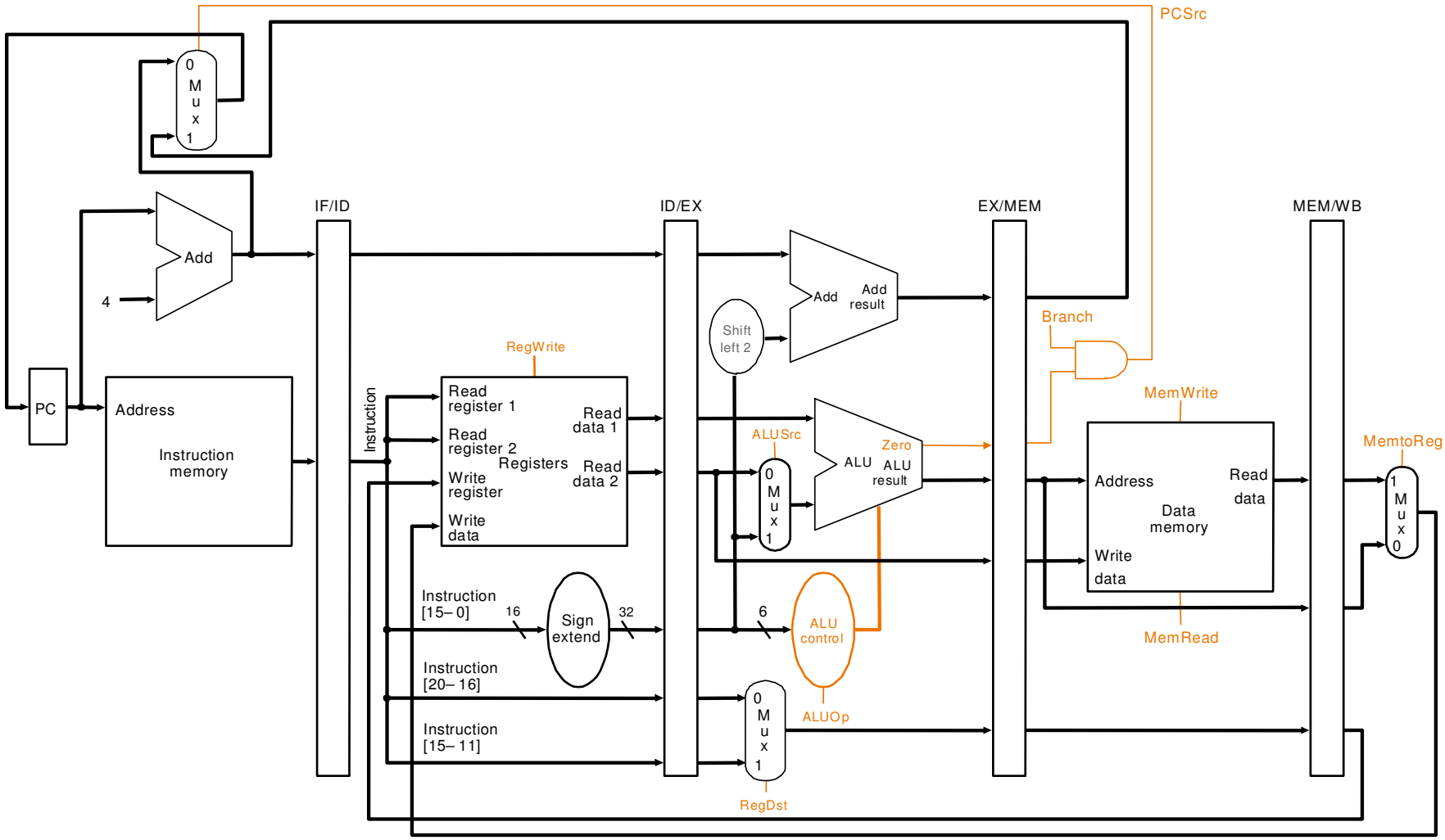








Controle do Pipeline



Controle do Pipeline

- 5 estágios. O que deve ser controlado em cada estágio?
  - 1º: Fetch da instrução e incremento do PC
  - 2º: Decodificação da instrução e Fetch dos registradores
  - 3º: Execução
  - 4º: Acesso à memória
  - 5º: Write back

Sinais de Controle

Instruction opcode	ALUOp	Instruction operation	Function code	Desired ALU action	ALU control input
LW	00	load word	XXXXXX	add	010
SW	00	store word	XXXXXX	add	010
Branch equal	01	branch equal	XXXXXX	subtract	110
R-type	10	add	100000	add	010
R-type	10	subtract	100010	subtract	110
R-type	10	AND	100100	and	000
R-type	10	OR	100101	or	001
R-type	10	set on less than	101010	set on less than	111

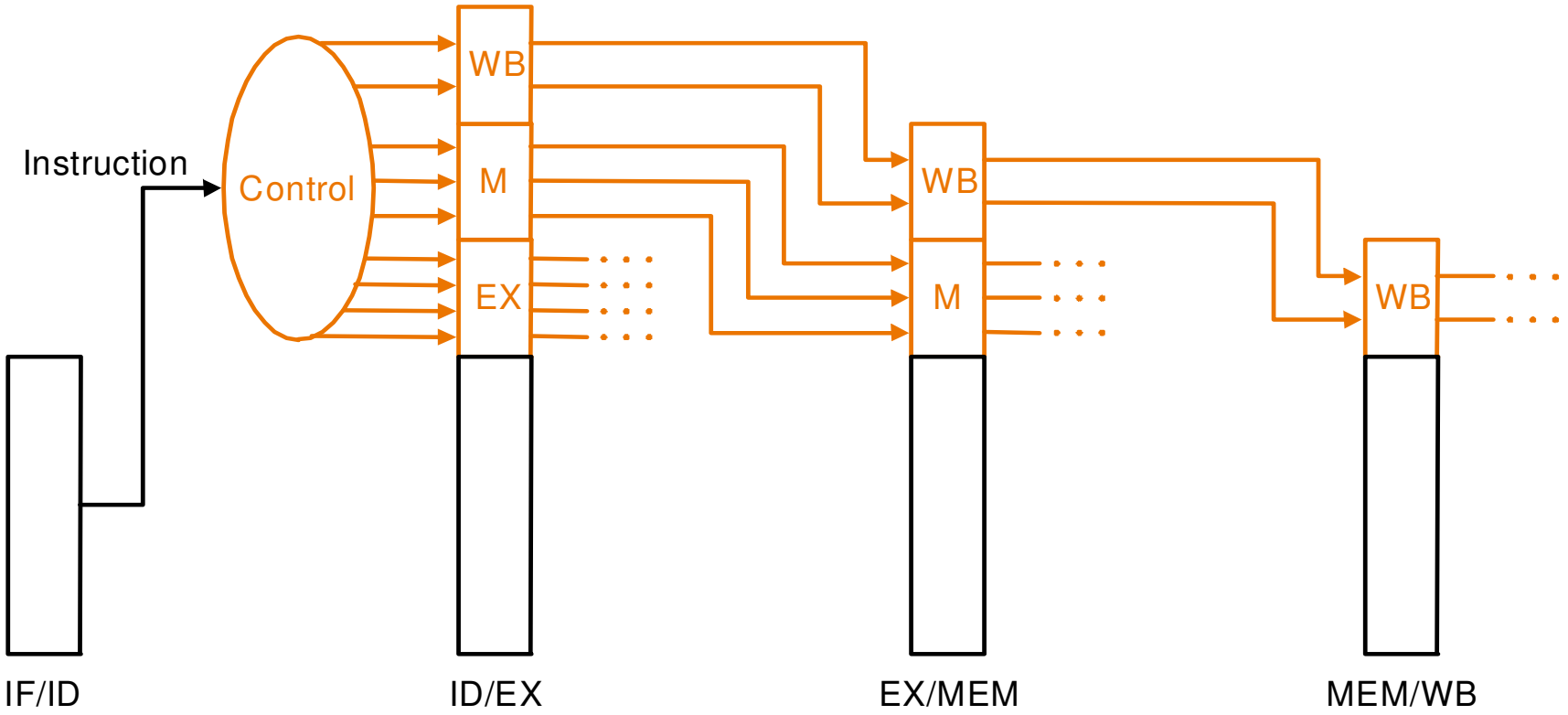
Sinais de Controle

Signal name	Effect when deasserted (0)	Effect when asserted (1)
RegDst	The register destination number for the Write register comes from the rt field (bits 20–16).	The register destination number for the Write register comes from the rd field (bits 15–11).
RegWrite	None	The register on the Write register input is written with the value on the Write data input.
ALUSrc	The second ALU operand comes from the second register file output (Read data 2).	The second ALU operand is the sign-extended, lower 16 bits of the instruction.
PCSrc	The PC is replaced by the output of the adder that computes the value of PC + 4.	The PC is replaced by the output of the adder that computes the branch target.
MemRead	None	Data memory contents designated by the address input are put on the Read data output.
MemWrite	None	Data memory contents designated by the address input are replaced by the value on the Write data input.
MemtoReg	The value fed to the register Write data input comes from the ALU.	The value fed to the register Write data input comes from the data memory.

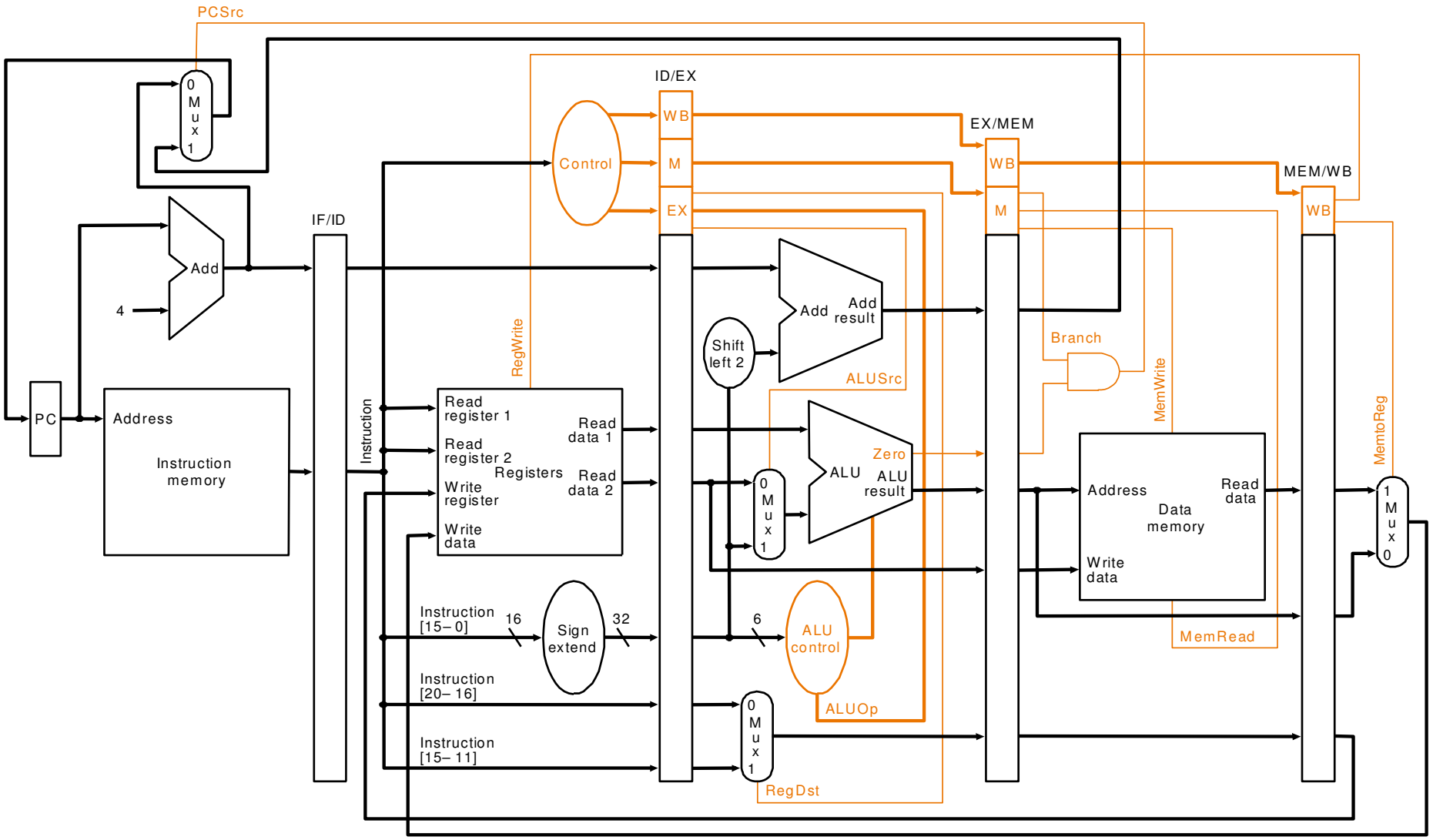
Sinais de Controle

	Execution/Address Calculation stage control lines				Memory access stage control lines			Write-back stage control lines	
Instruction	Reg Dst	ALU Op1	ALU Op0	ALU Src	Branch	Mem Read	Mem Write	Reg write	Mem to Reg
R-format	1	1	0	0	0	0	0	1	0
`lw`	0	0	0	1	0	1	0	1	1
`sw`	X	0	0	1	0	0	1	0	X
`beq`	X	0	1	0	1	0	0	0	X

Unidade de Controle

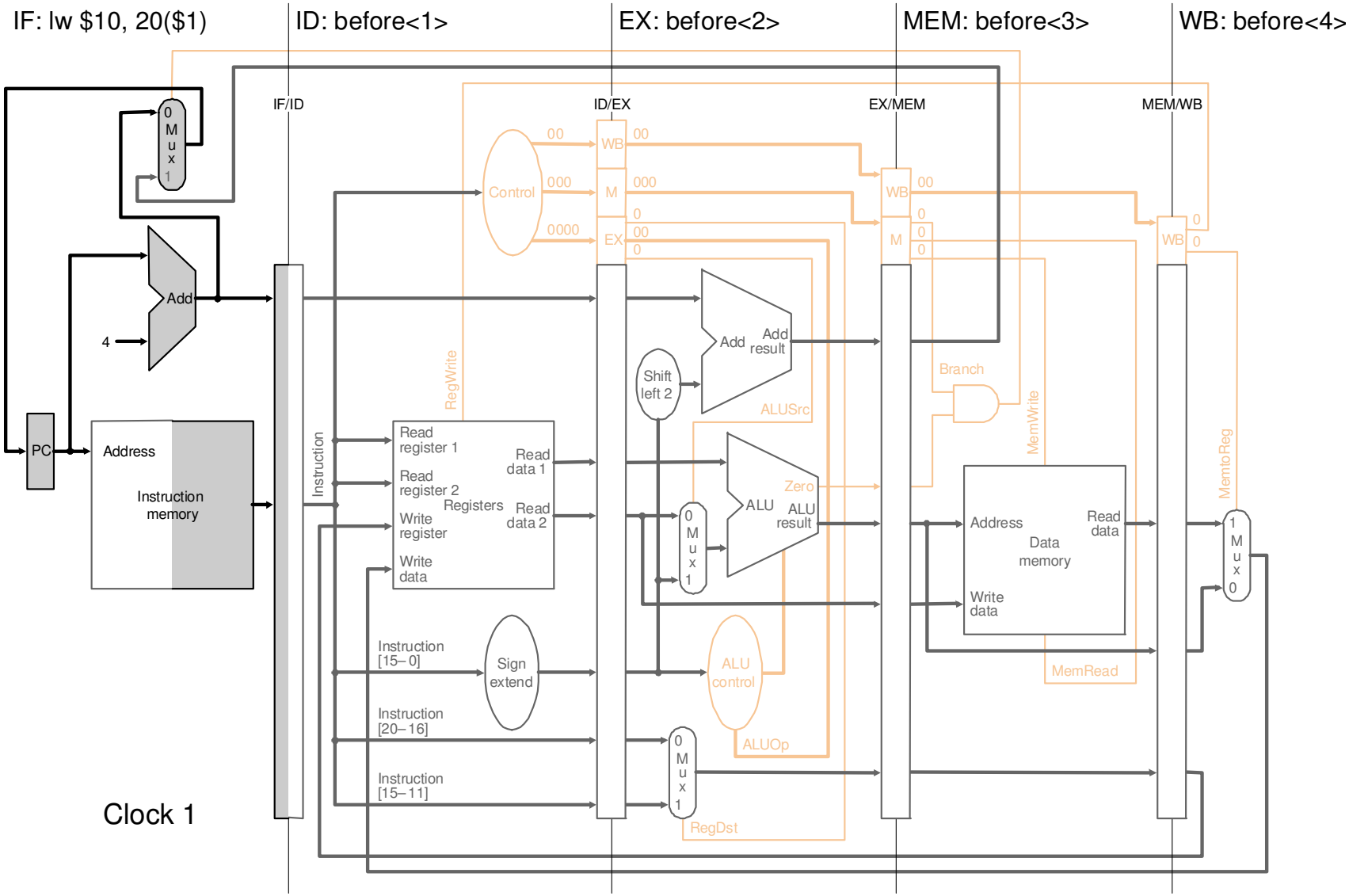


Datapath com Controle



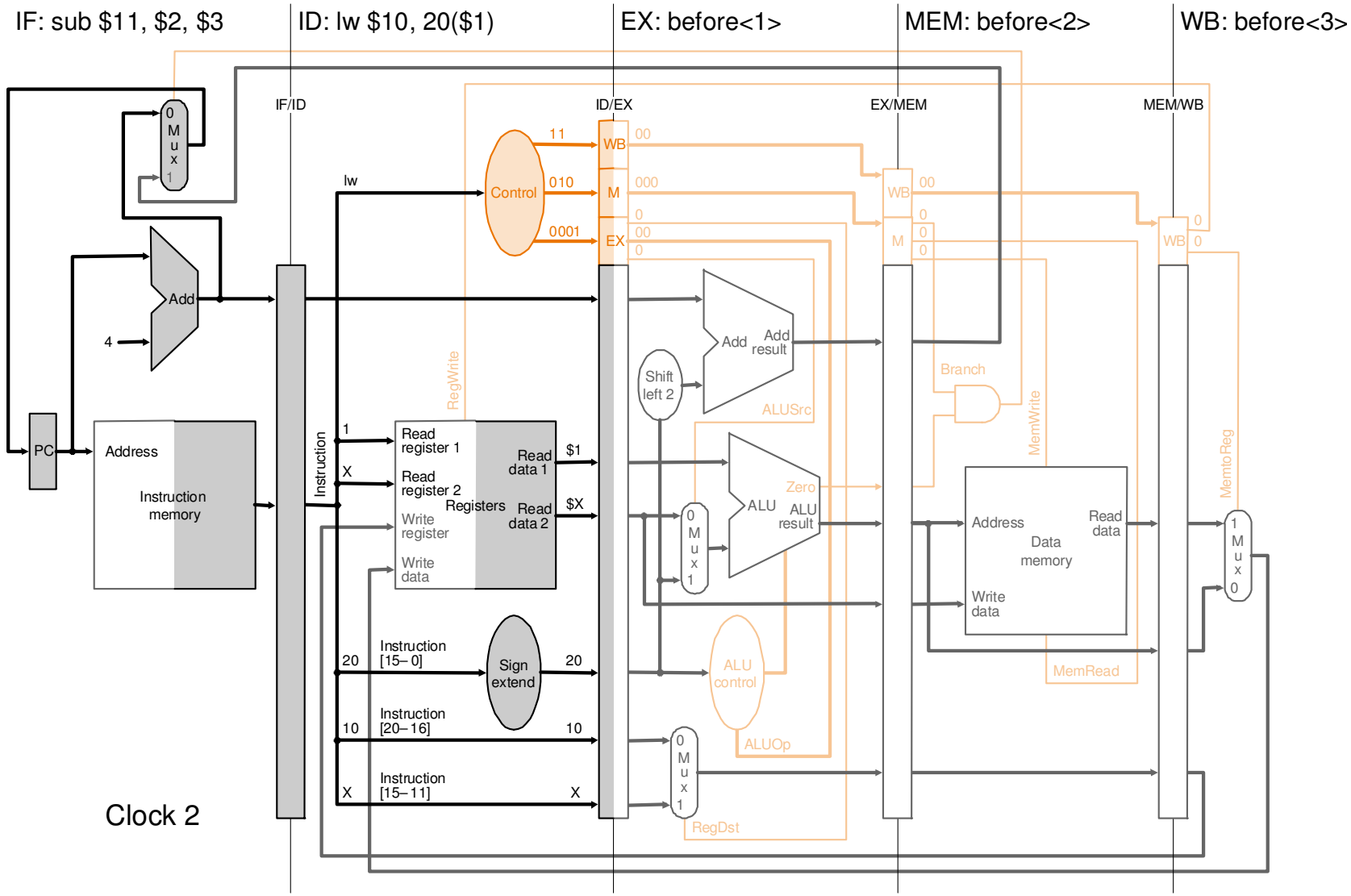
lw $10, 20 ($1)
sub $11, $2, $3
and $12, $4, $5
or $13, $6, $7
add $14, $8, $9

1º ciclo



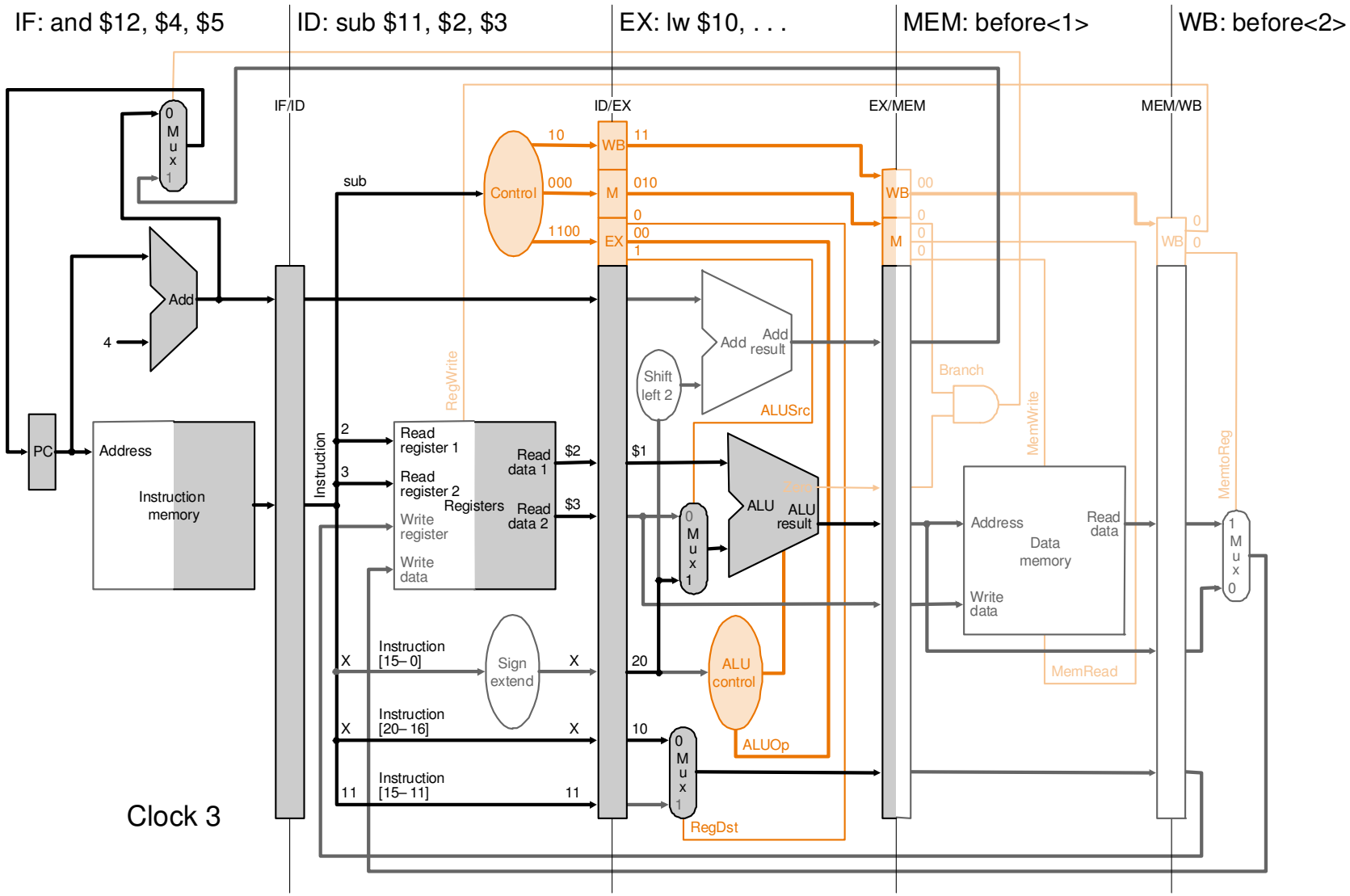
```
lw    $10, 20 ($1)
sub   $11, $2,  $3
and   $12, $4,  $5
or    $13, $6,  $7
add   $14, $8,  $9
```

2º ciclo



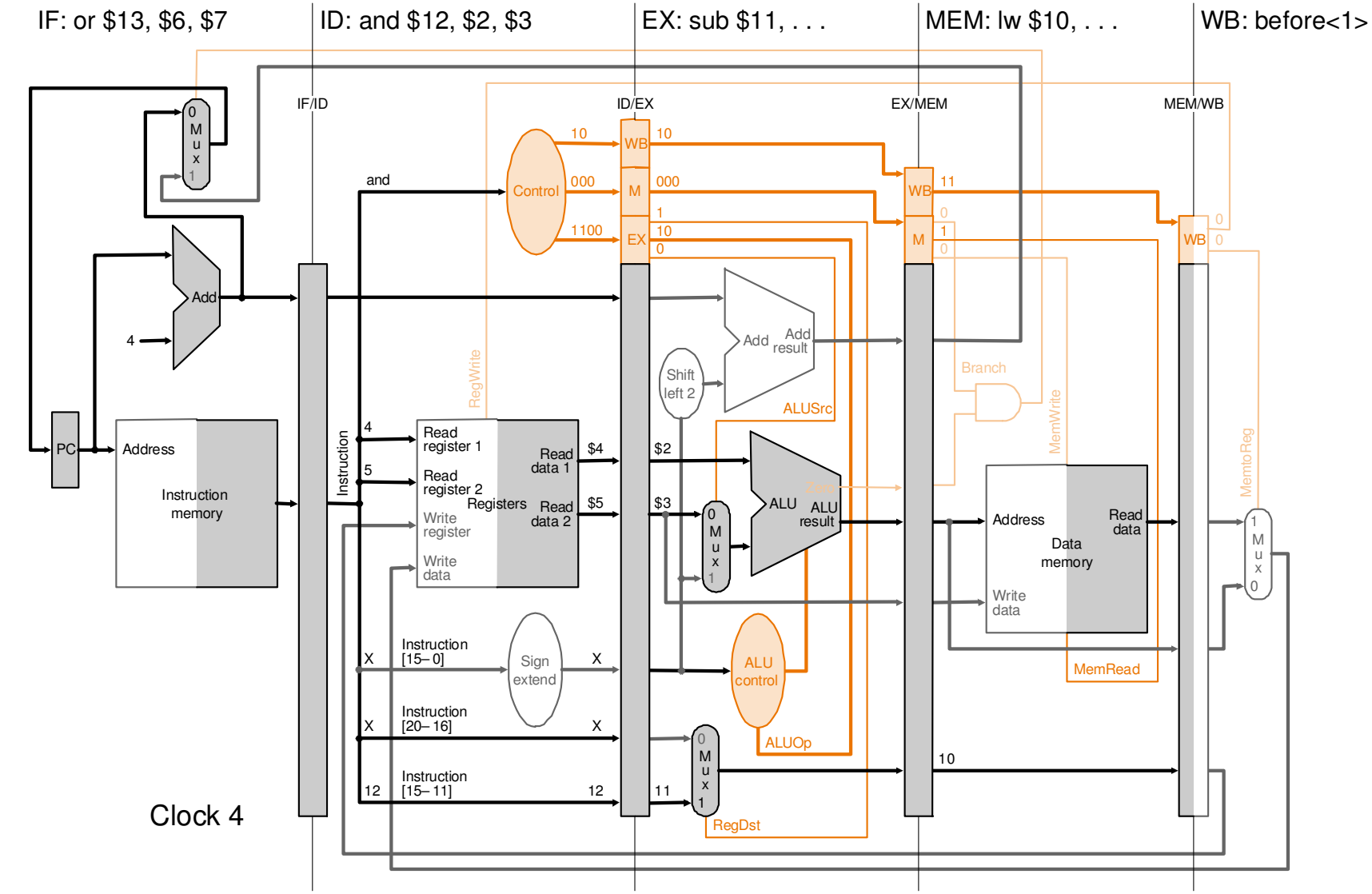
```
lw    $10, 20 ($1)
sub   $11, $2,  $3
and   $12, $4,  $5
or    $13, $6,  $7
add   $14, $8,  $9
```

3º ciclo



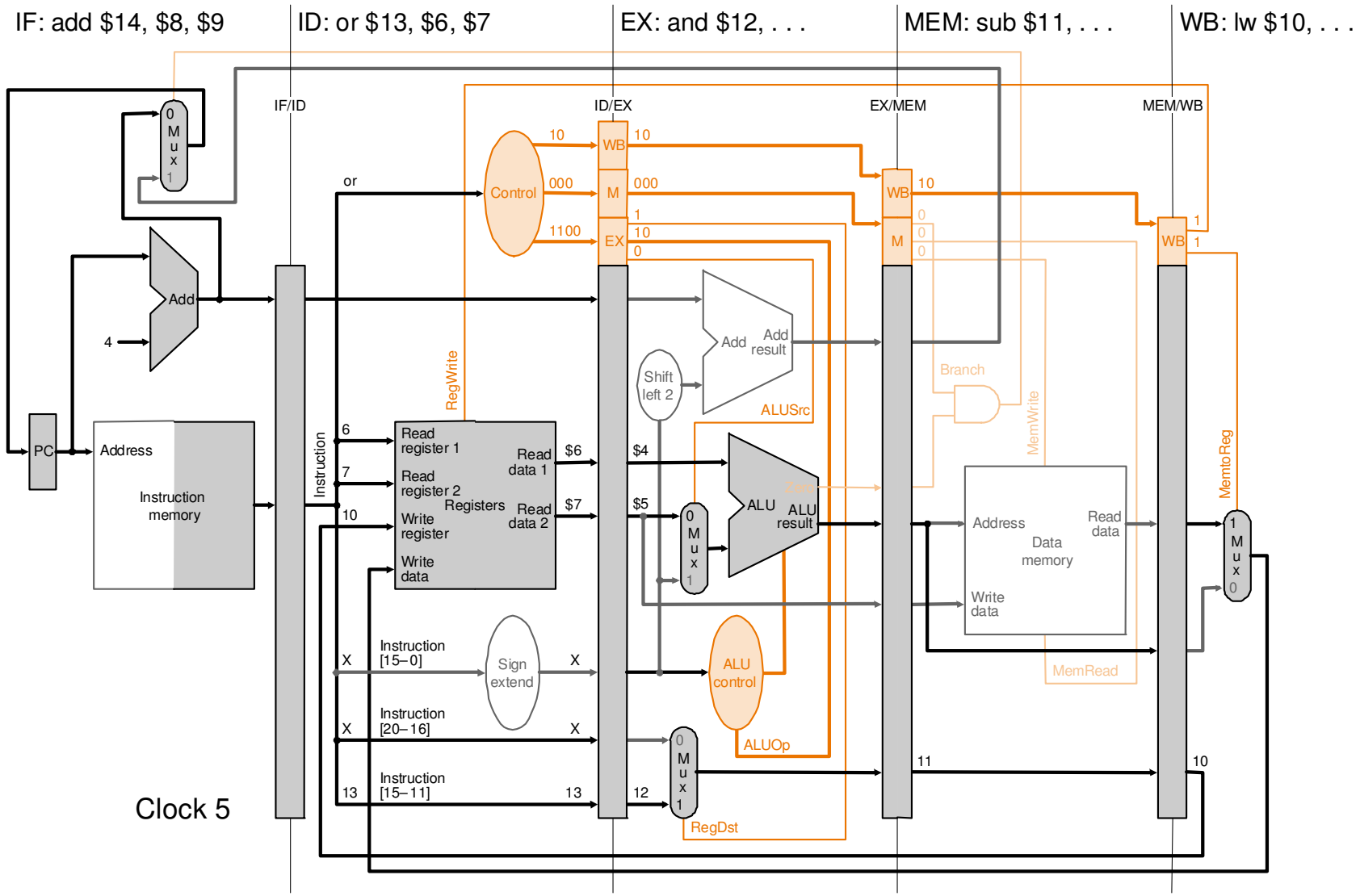
```
lw    $10, 20 ($1)
sub   $11, $2,  $3
and   $12, $4,  $5
or    $13, $6,  $7
add   $14, $8,  $9
```

4º ciclo



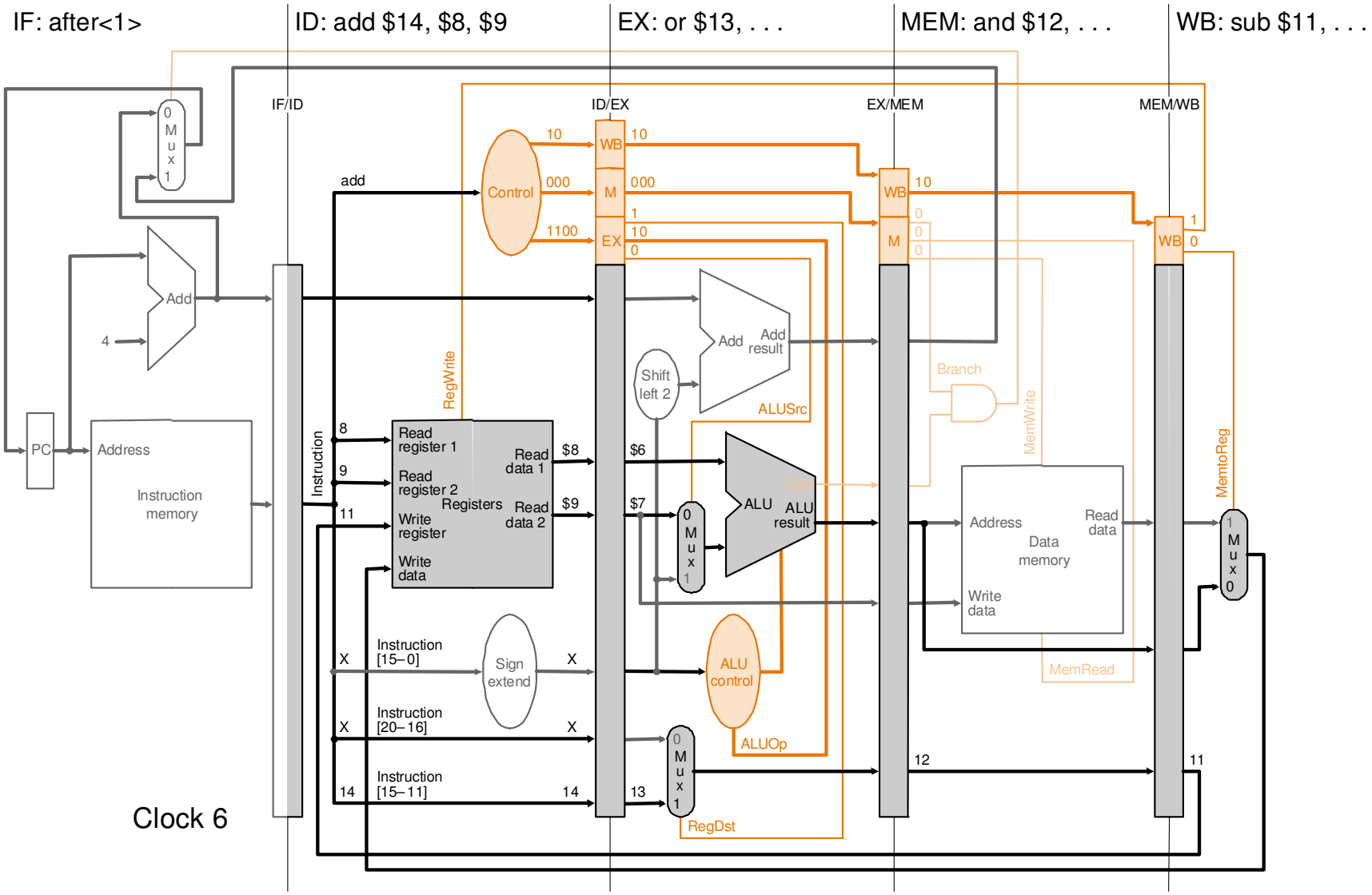
lw $10, 20 ($1)
sub $11, $2, $3
and $12, $4, $5
or $13, $6, $7
add $14, $8, $9

5º ciclo



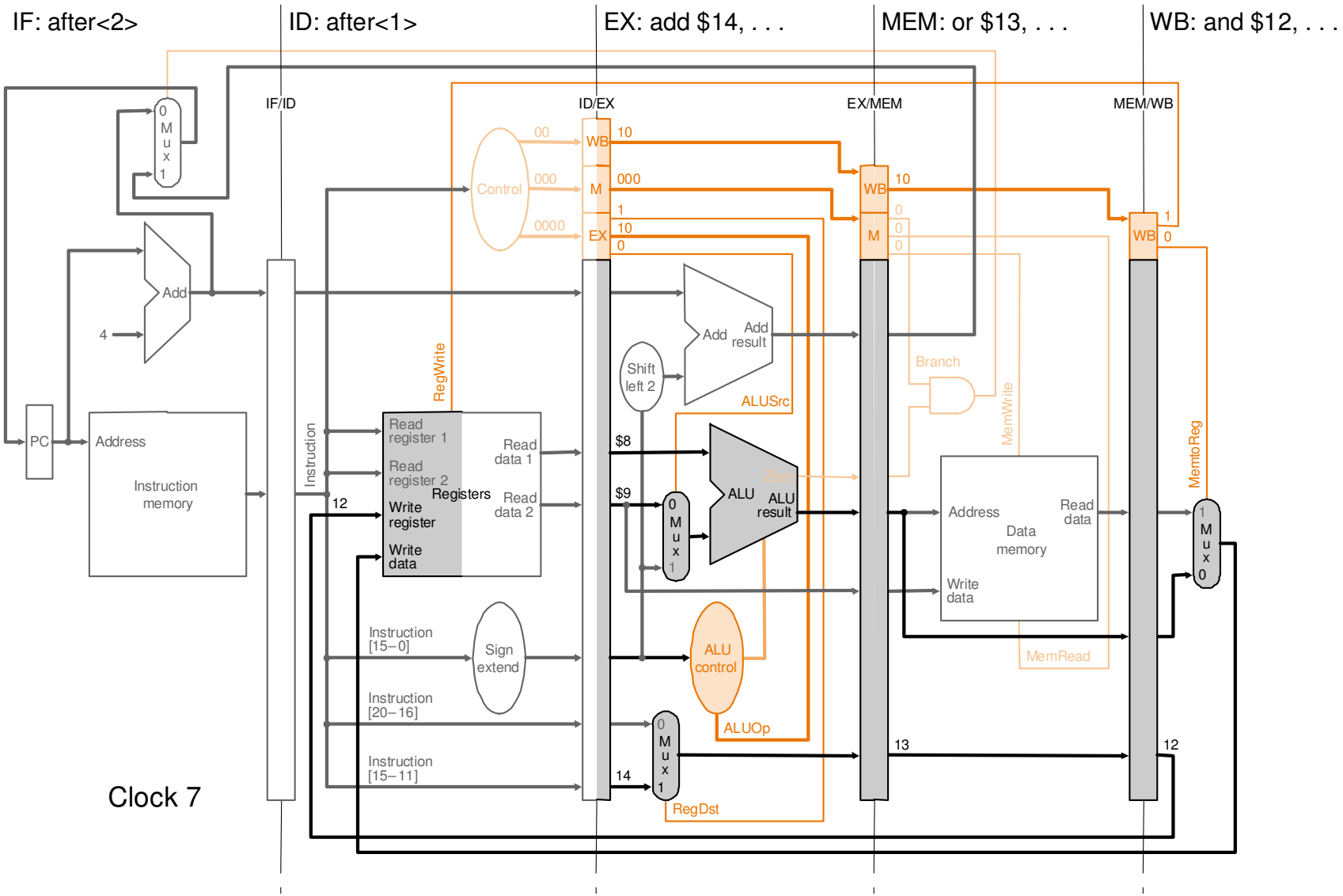
lw $10, 20 ($1)
sub $11, $2, $3
and $12, $4, $5
or $13, $6, $7
add $14, $8, $9

6º ciclo



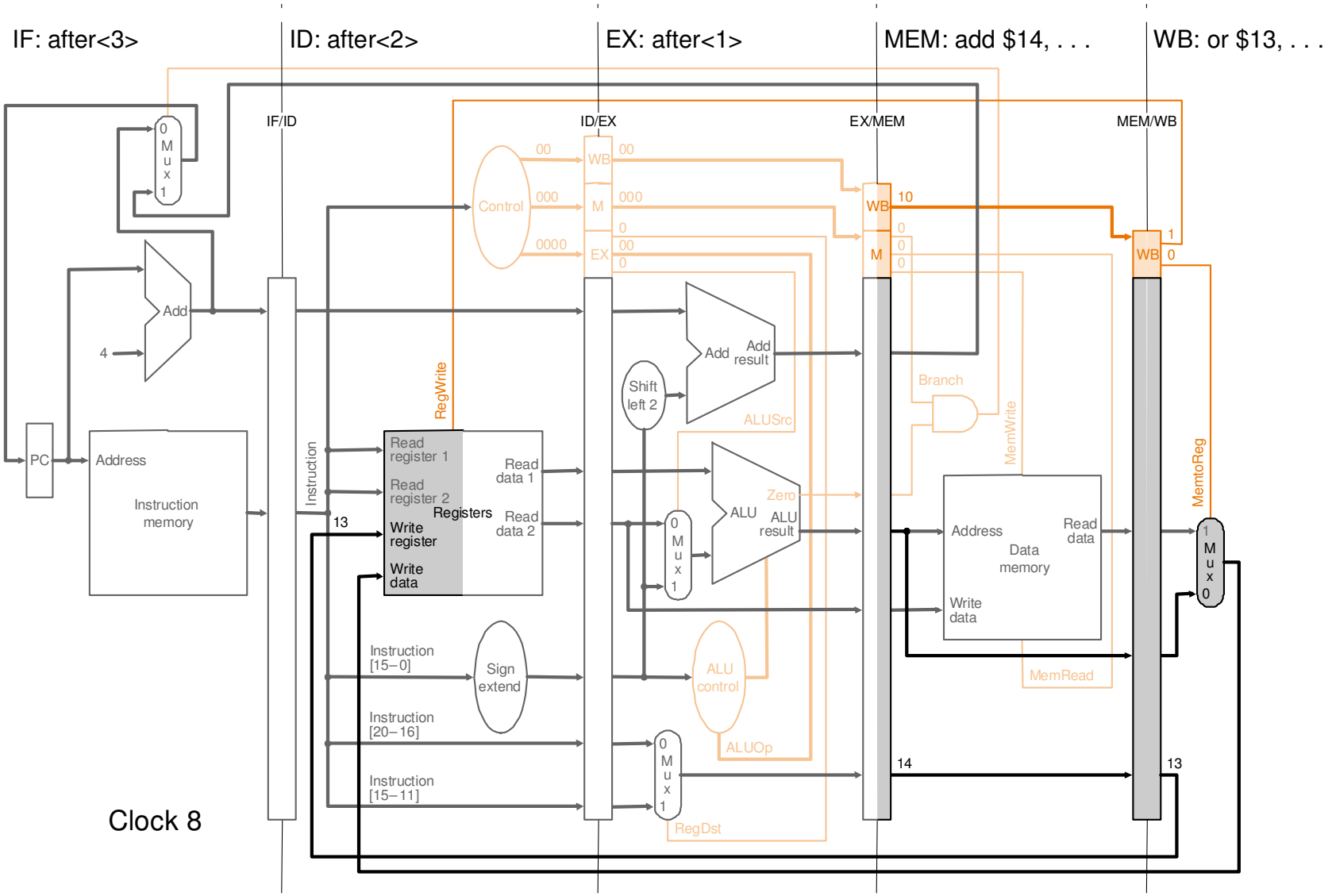
lw $10, 20 ($1)
sub $11, $2, $3
and $12, $4, $5
or $13, $6, $7
add $14, $8, $9

7º ciclo



```
lw    $10, 20 ($1)
sub   $11, $2,  $3
and   $12, $4,  $5
or    $13, $6,  $7
add   $14, $8,  $9
```

8º ciclo



```
lw    $10, 20 ($1)
sub   $11, $2,  $3
and   $12, $4,  $5
or    $13, $6,  $7
add   $14, $8,  $9
```

9º ciclo