IC-UNICAMP

MO401

IC/Unicamp

2013s1

Prof Mario Côrtes

Capítulo 0: Revisão

IC-UNICAMP

Tópicos de revisão

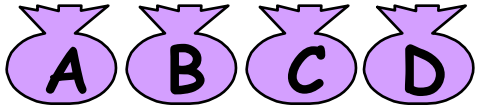
- Revisão:
  - Pipelining
  - Desempenho
  - Hierarquia de Memórias (cache)
  - ISA MIPS64
- Referências
  - Livro Texto: “Computer Architecture: A Quantitative Approach” – 5th edition, Hennessy & Patterson
  - Revisão: “Computer Organization and Design the hardware / software interface” , Patterson & Hennessy
  - material: Prof. Paulo Centoducatte

Revisão: Pipeline

Pipelining - Conceito

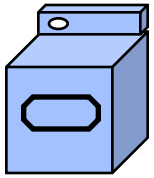
- Exemplo: Lavanderia

- 4 trouxas para serem lavadas



  - Lavar: 30 minutos

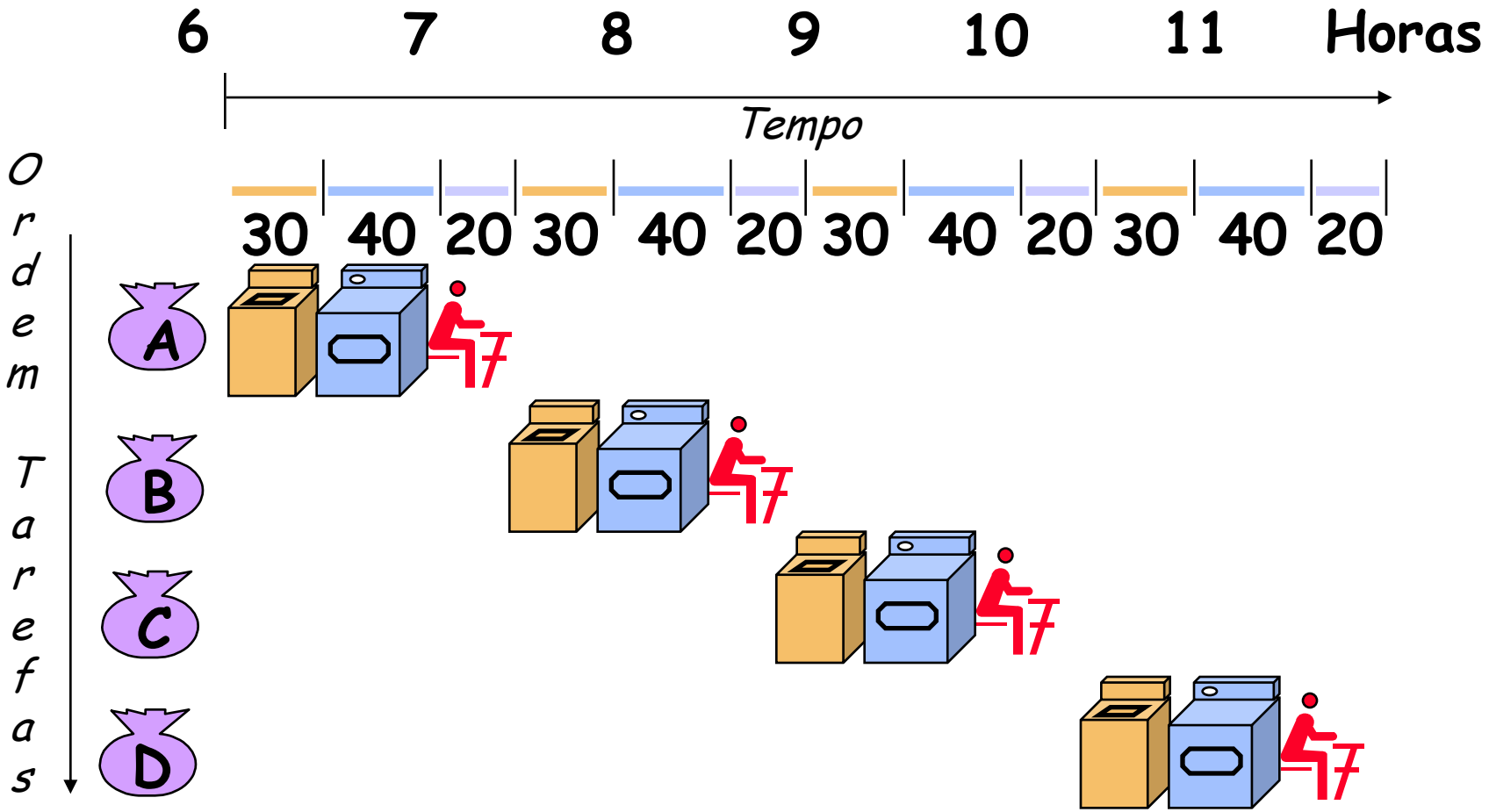
  - Secar: 40 minutos



  - Passar: 20 minutos

IC-UNICAMP

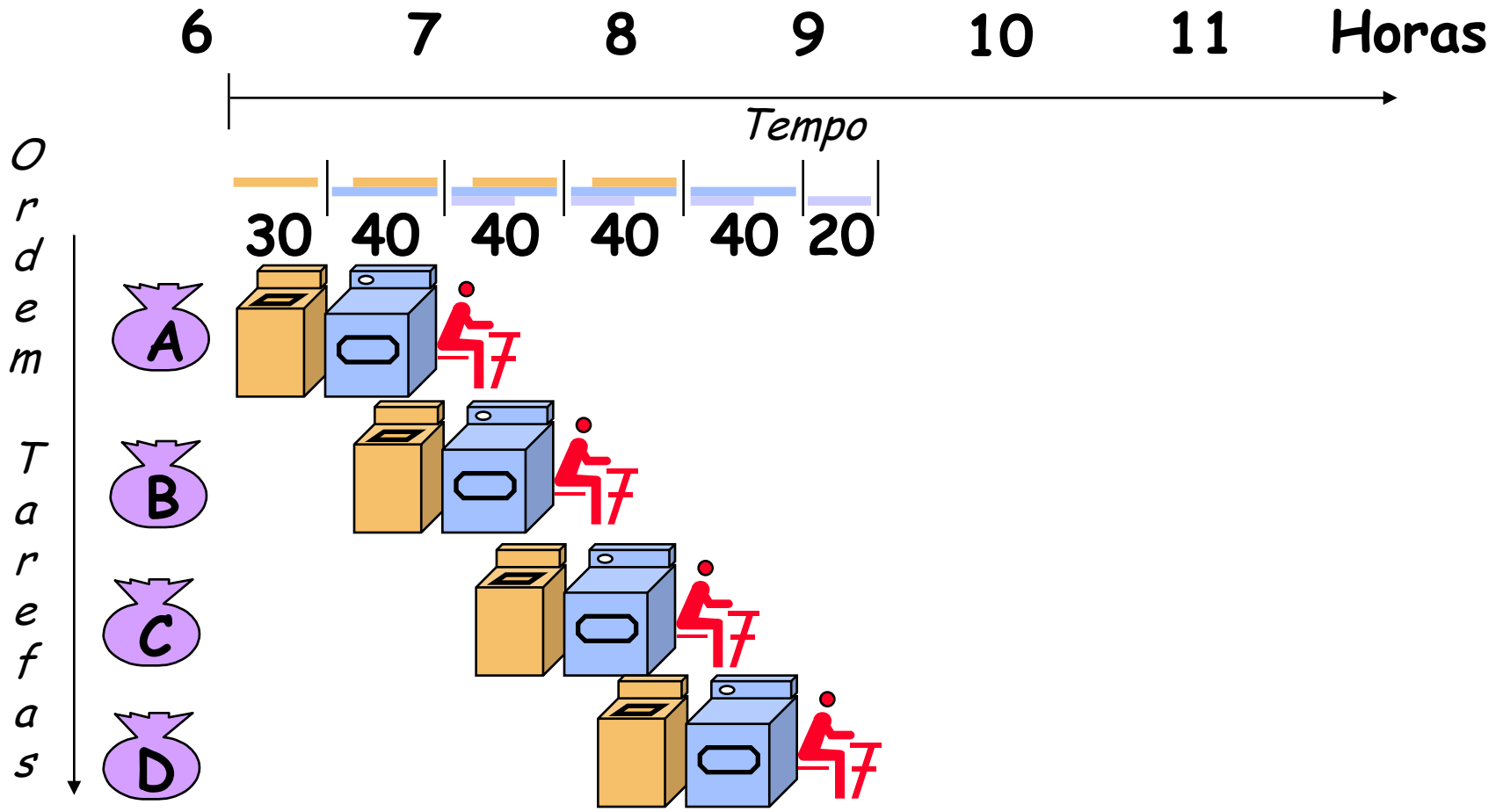
Lavanderia Seqüencial



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

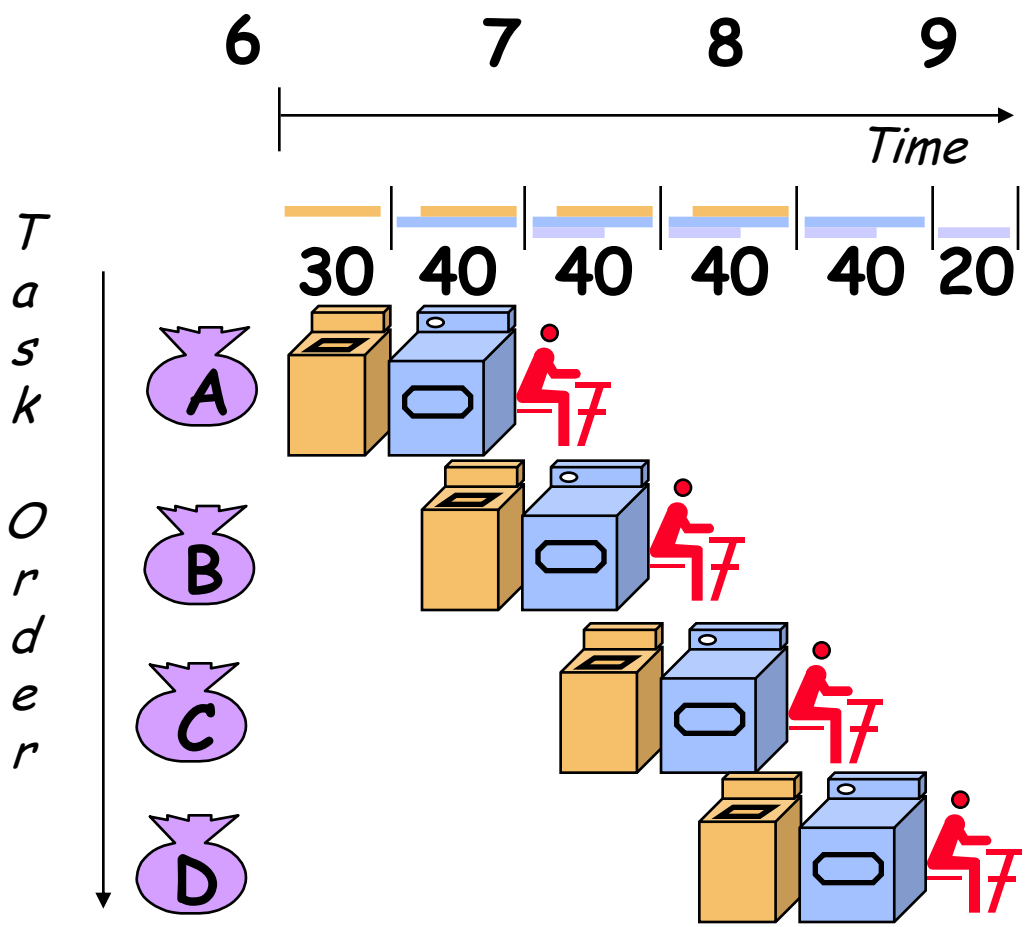
Lavanderia Pipelined

IC-UNICAMP



- 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
- Múltiplas 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

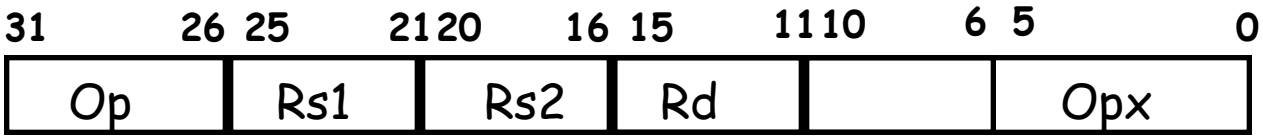
- Executam bilhões de instruções: throughput
- Características desejáveis em um conjunto de instruções (ISA) para pipelining?
  - Instruções de tamanho variável vs. Todas instruções do mesmo tamanho?
  - Operandos em memória em qq operações vs. operandos em memória somente para loads e stores?
  - Formato das instruções irregular vs. formato regular das instruções (ié. Operandos nos mesmos lugares)?

Um RISC Típico

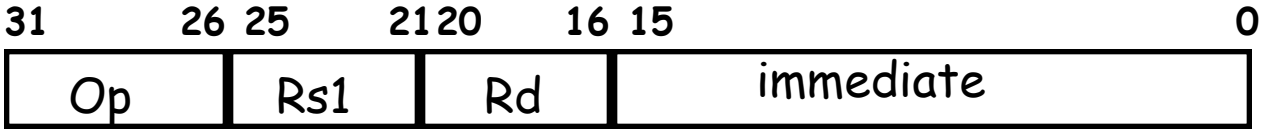
- Formato de instruções de 32-bit (3 formatos)
- Acesso à memória somente via load/store
- 32 GPR de 32-bits (R0 contém zero)
- Instruções aritméticas: 3-address, reg-reg, registradores no mesmo lugar
- Modo de endereçamento para load/store (base + displacement). Sem indireção
- Condições simples para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

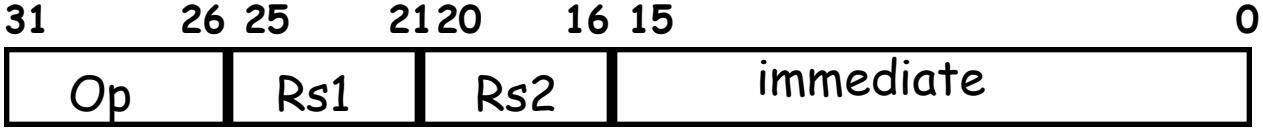
Register-Register



Register-Immediate



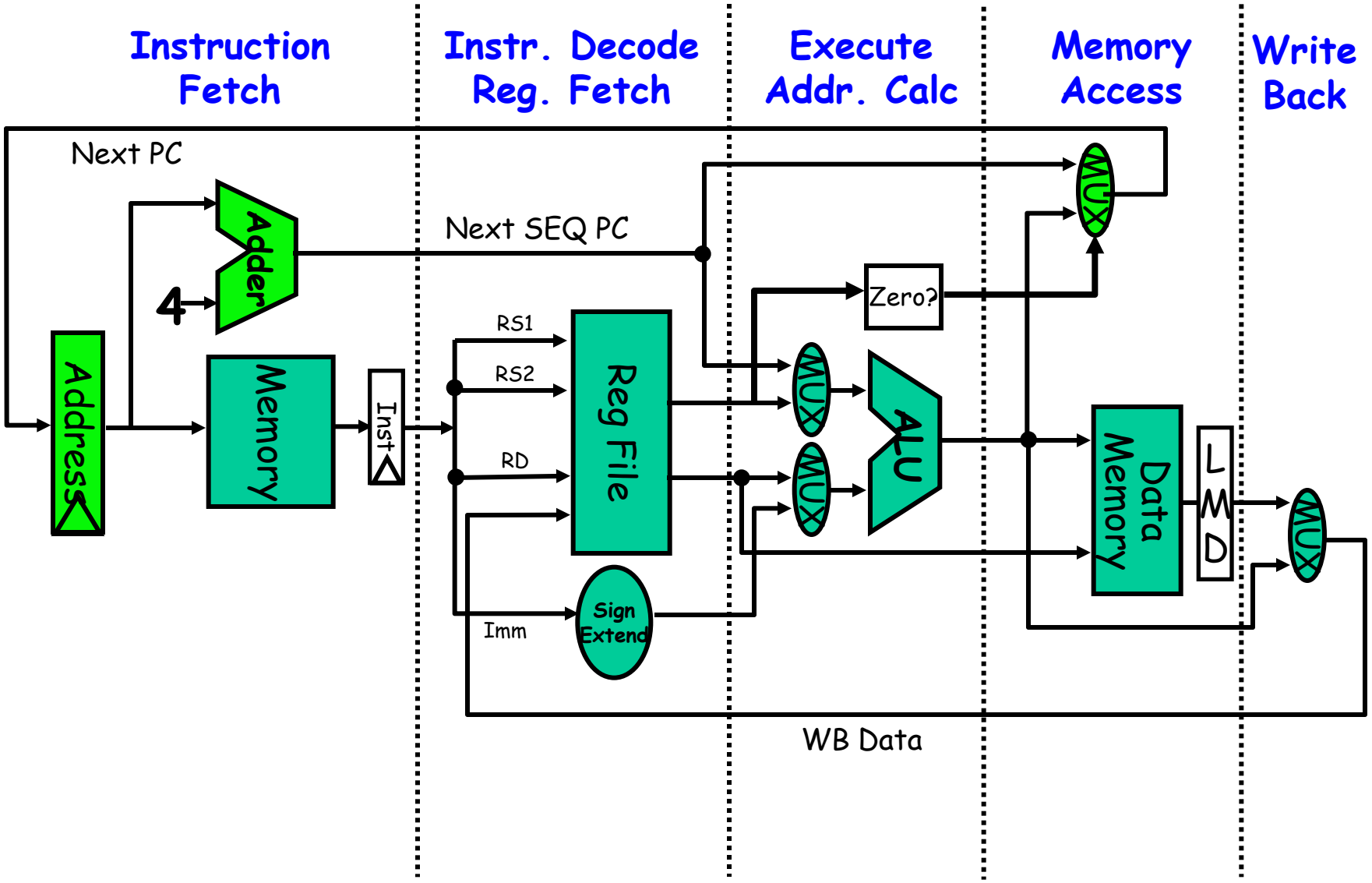
Branch



Jump / Call

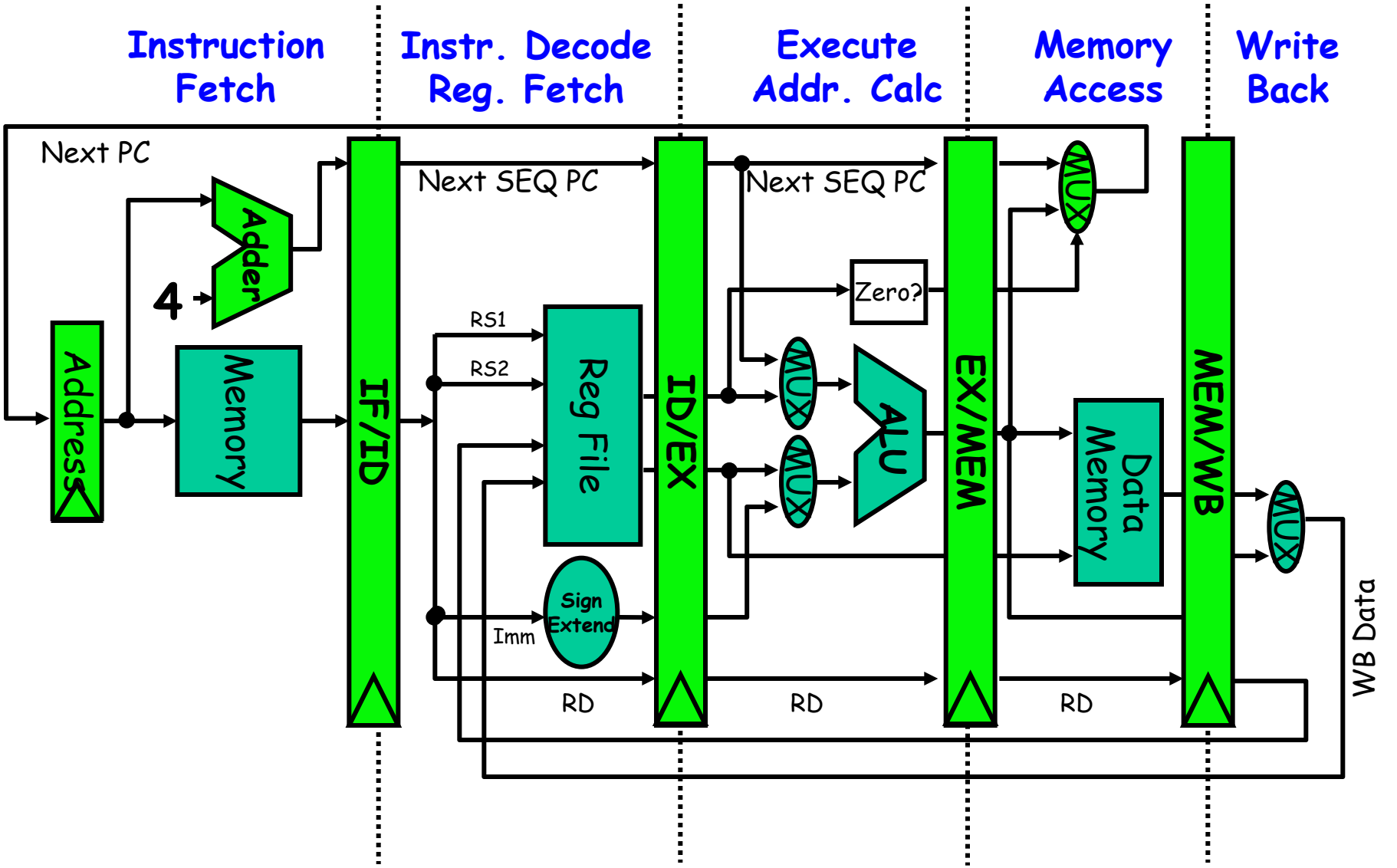
MIPS Datapath - 5 estágios

IC-UNICAMP



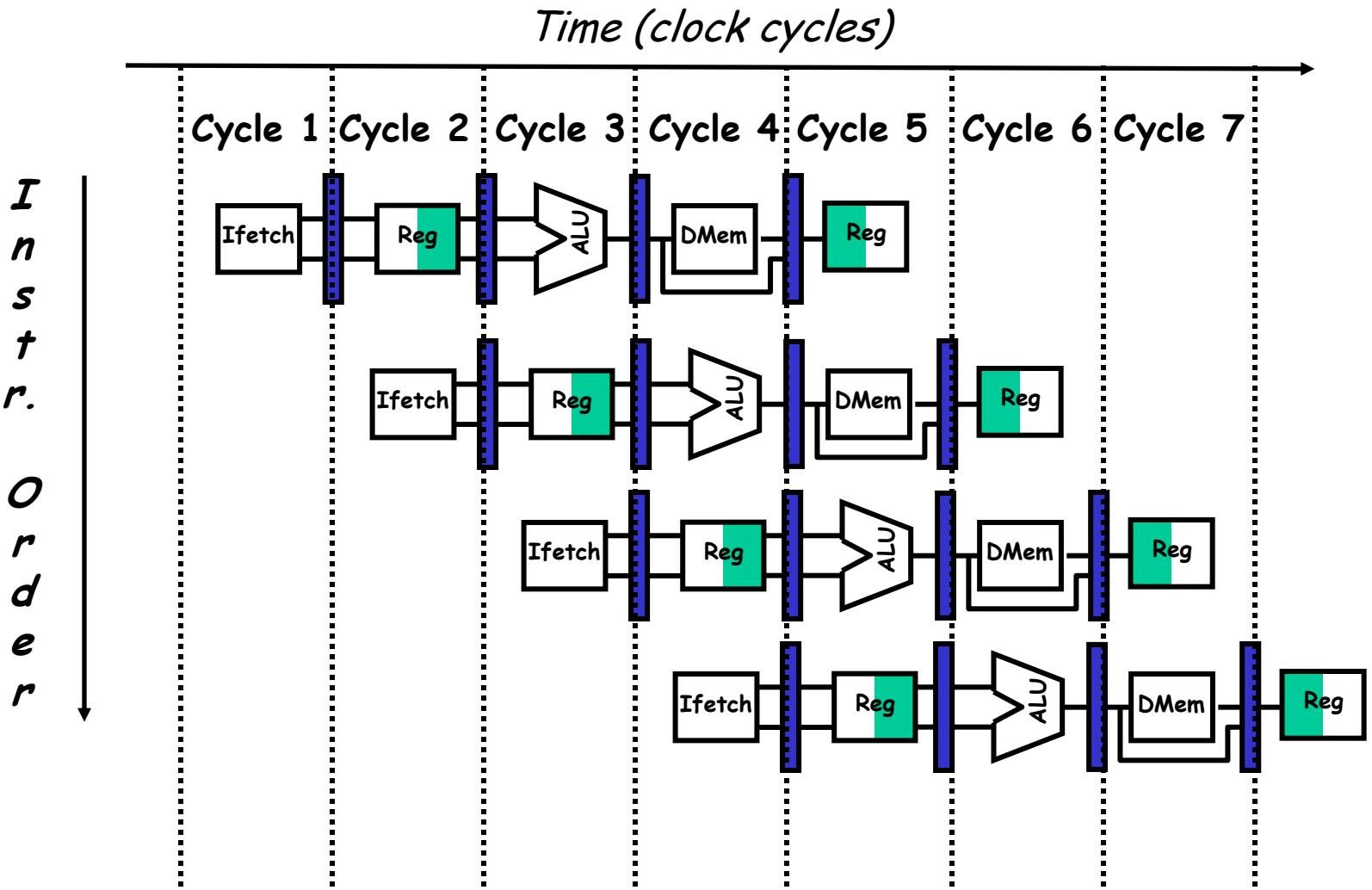
MIPS Datapath - 5 estágios

IC-UNICAMP



Pipelining

IC-UNICAMP

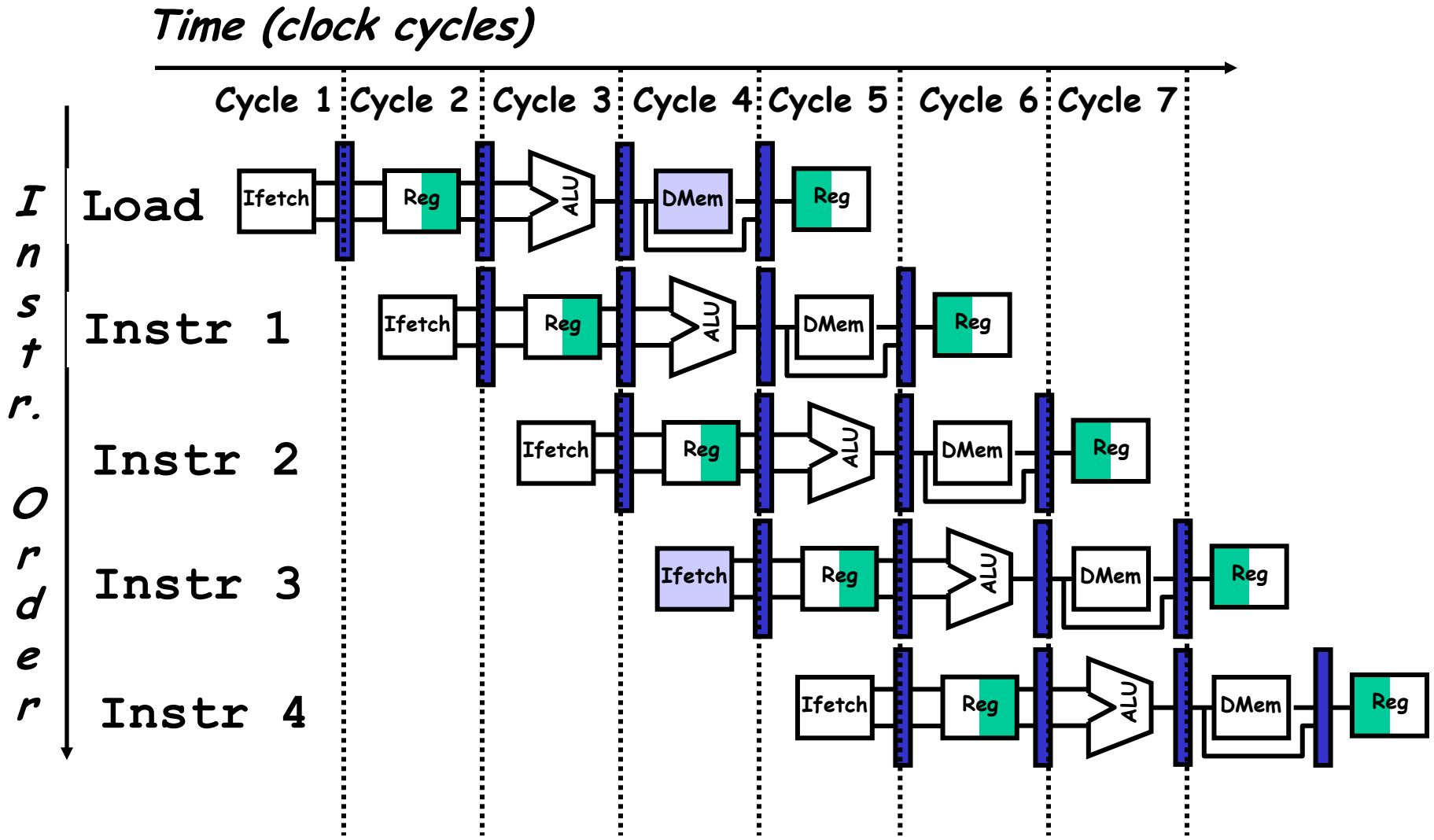


Limites de Pipelining

- 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).

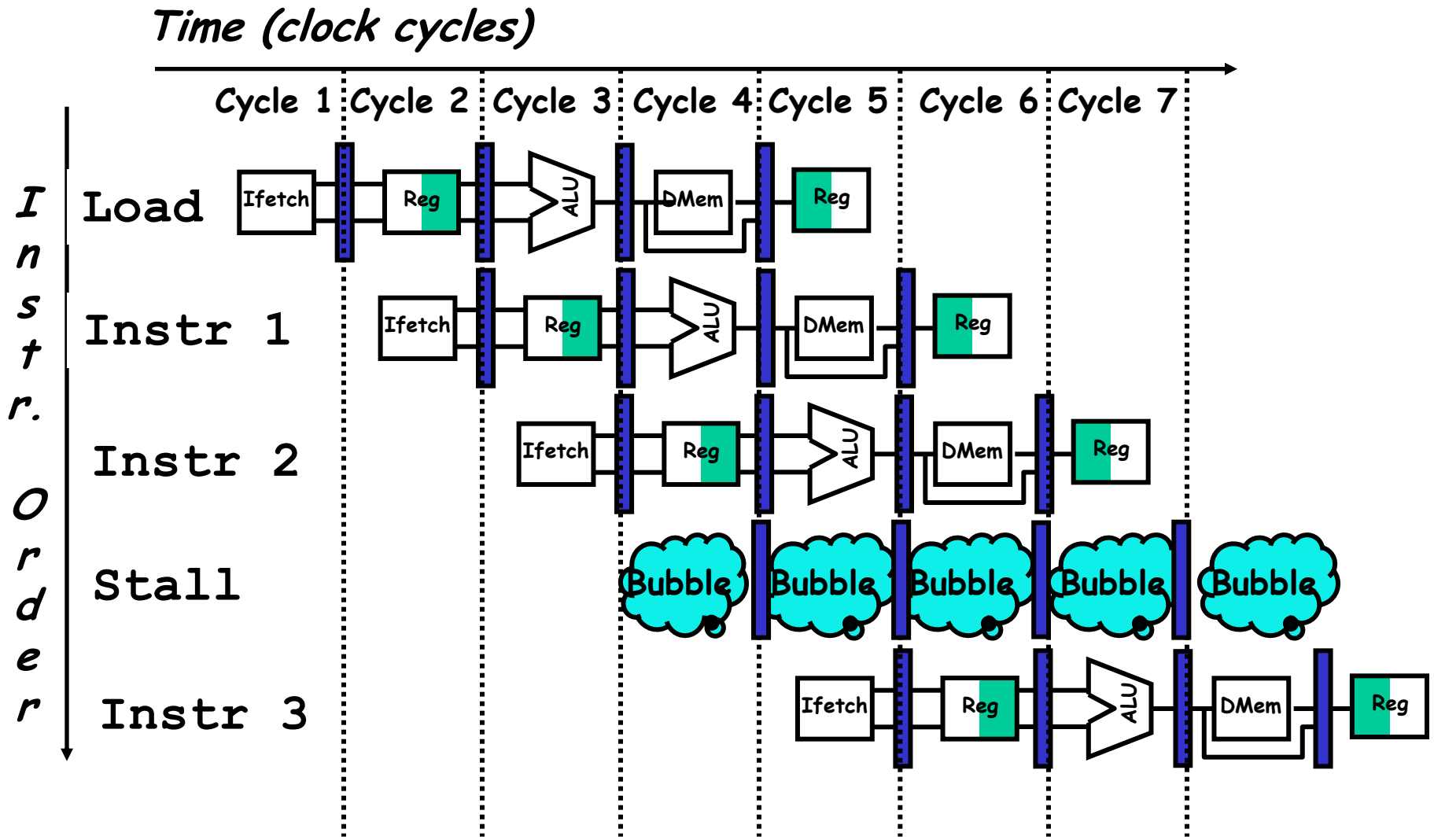
Memória Única (D/I) - Structural Hazards

IC-UNICAMP



Memória Única (D/I) - Structural Hazards

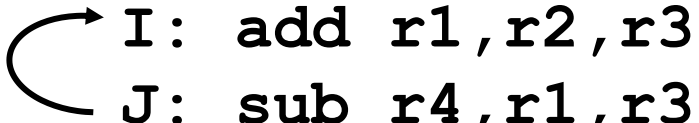
IC-UNICAMP



Data Hazards

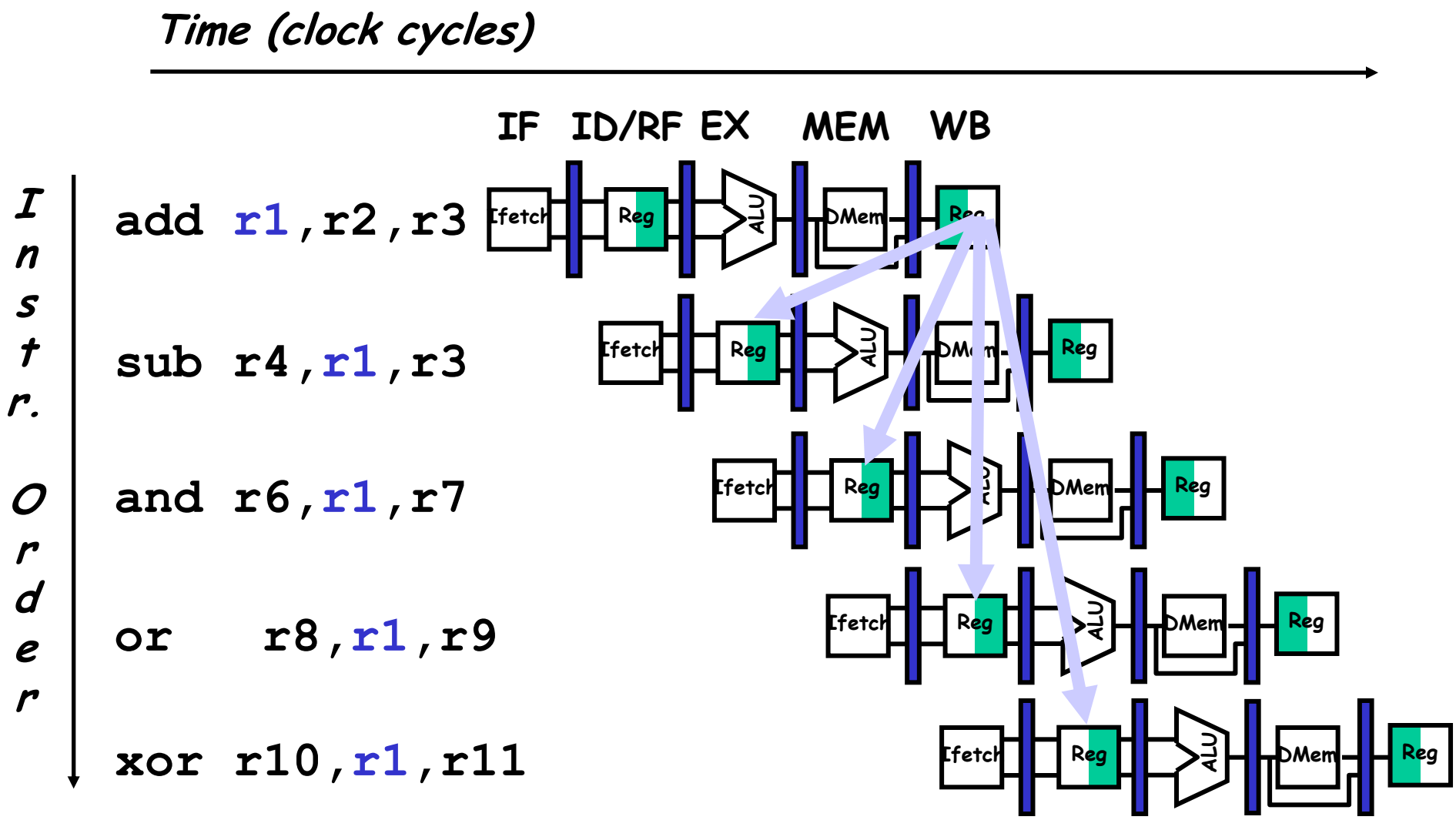
- Read After Write (RAW)

  $Instr_J$ lê o operando antes da $Instr_I$ escreve-lo



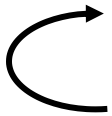
  – Causada por uma “Dependência” (nomenclatura de compiladores).

Data Hazard em R1



Data Hazards

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

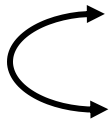


```
I: sub r4,r1,r3
J: add r1,r2,r3
K: mul r6,r1,r7
```

- 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 Hazards

- Write After Write (WAW)
  InstrJ escreve o operando antes que a InstrI 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 pipelines mais sofisticados)

Data Hazards – Solução por SW

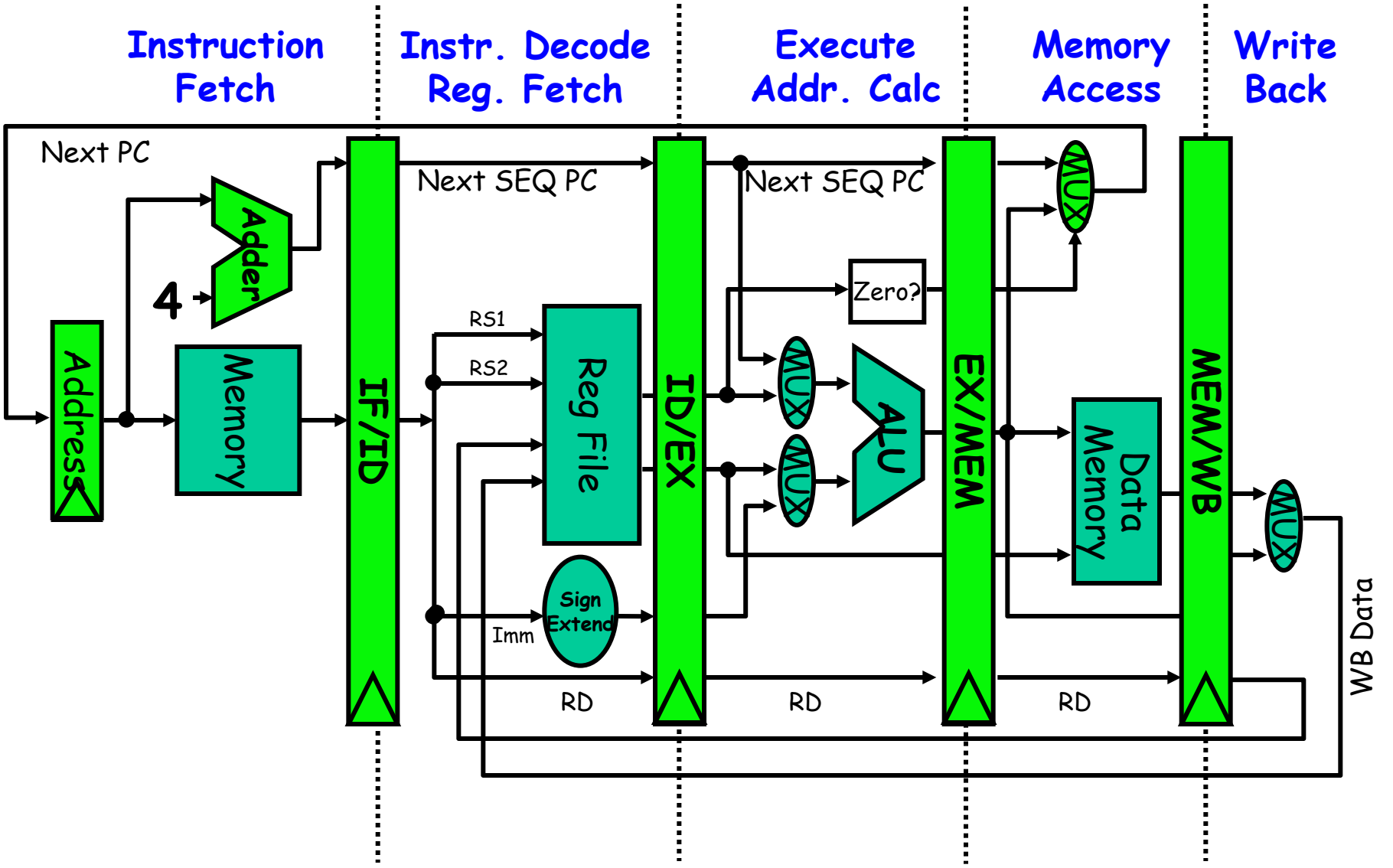
- Compilador reconhece o data hazard e troca a ordem das instruções (quando possível)
- Compilador reconhece o data hazard e adiciona nops
- Exemplo:

```
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

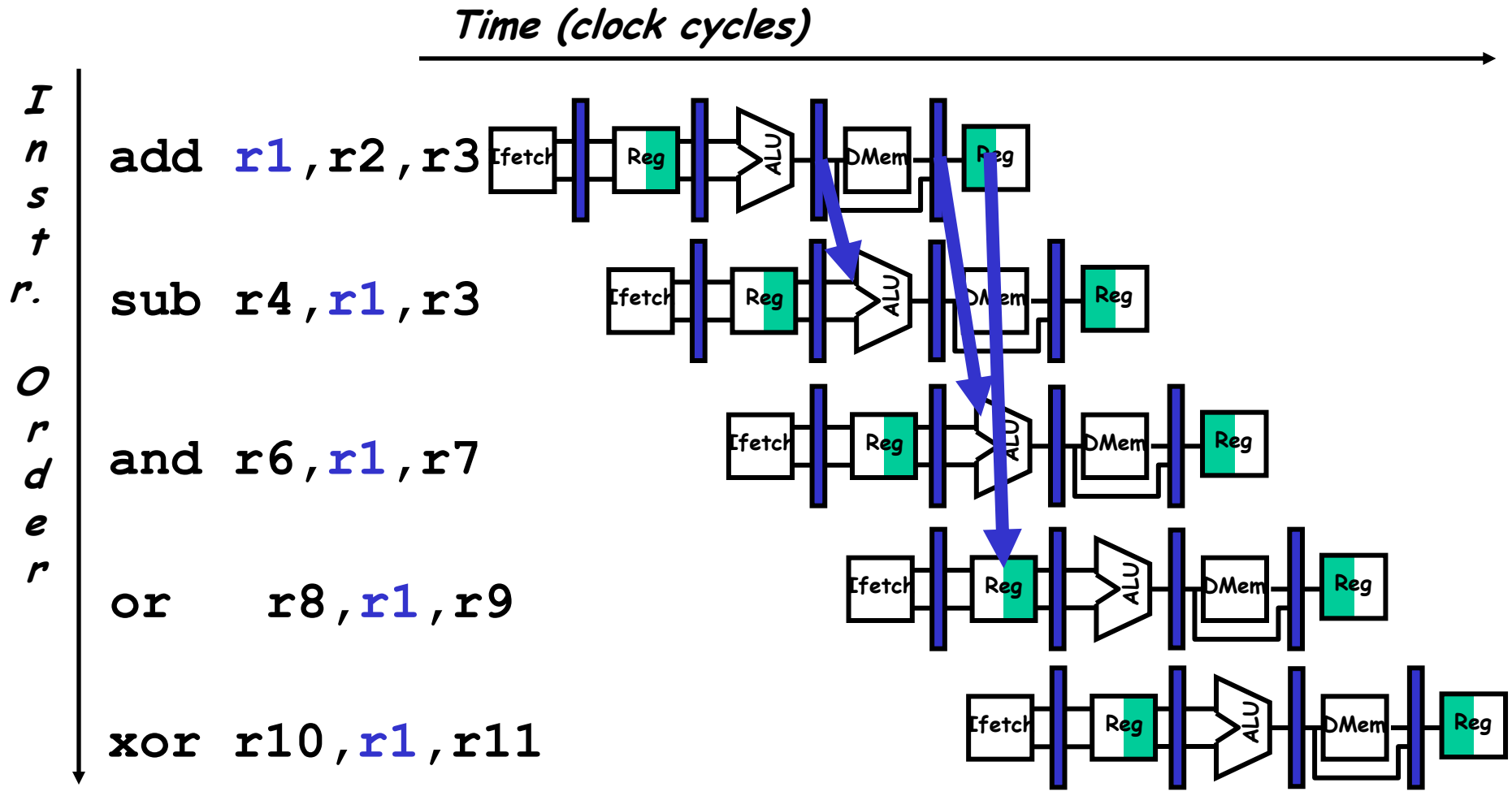
- Hazard ocorre quando a instr. Lê (no estágio ID) um reg que será escrito, por uma instr. anterior (no estágio EX, MEM, WB)
- Solução: Detectar o hazard e parar a instrução no pipeline até o hazard ser resolvido
- Detectar o hazard pela comparação do campo read no IF/ID pipeline register com o campo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

MIPS Datapath - 5 estágios

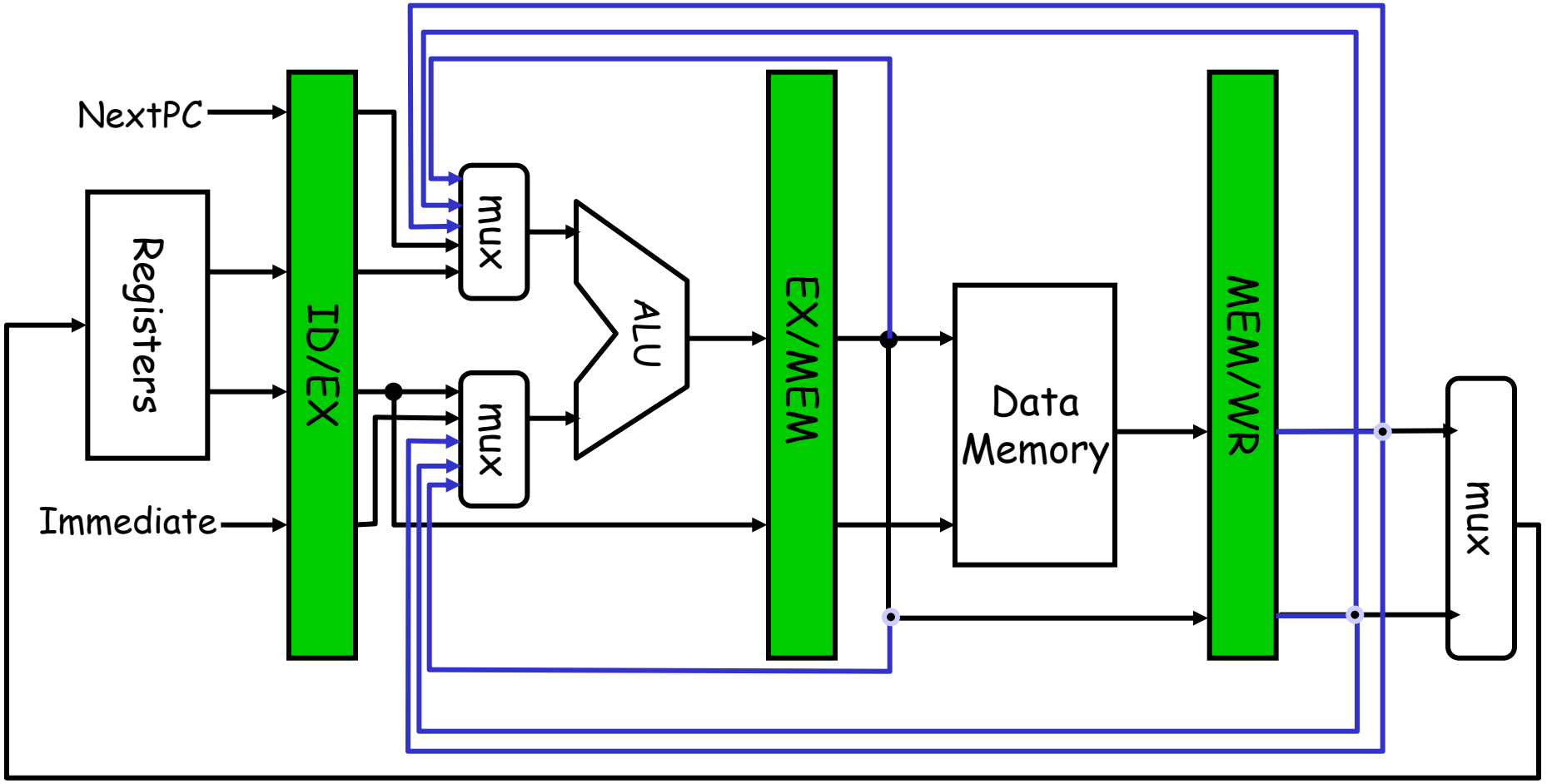


Redução do Data Hazard - Forwarding

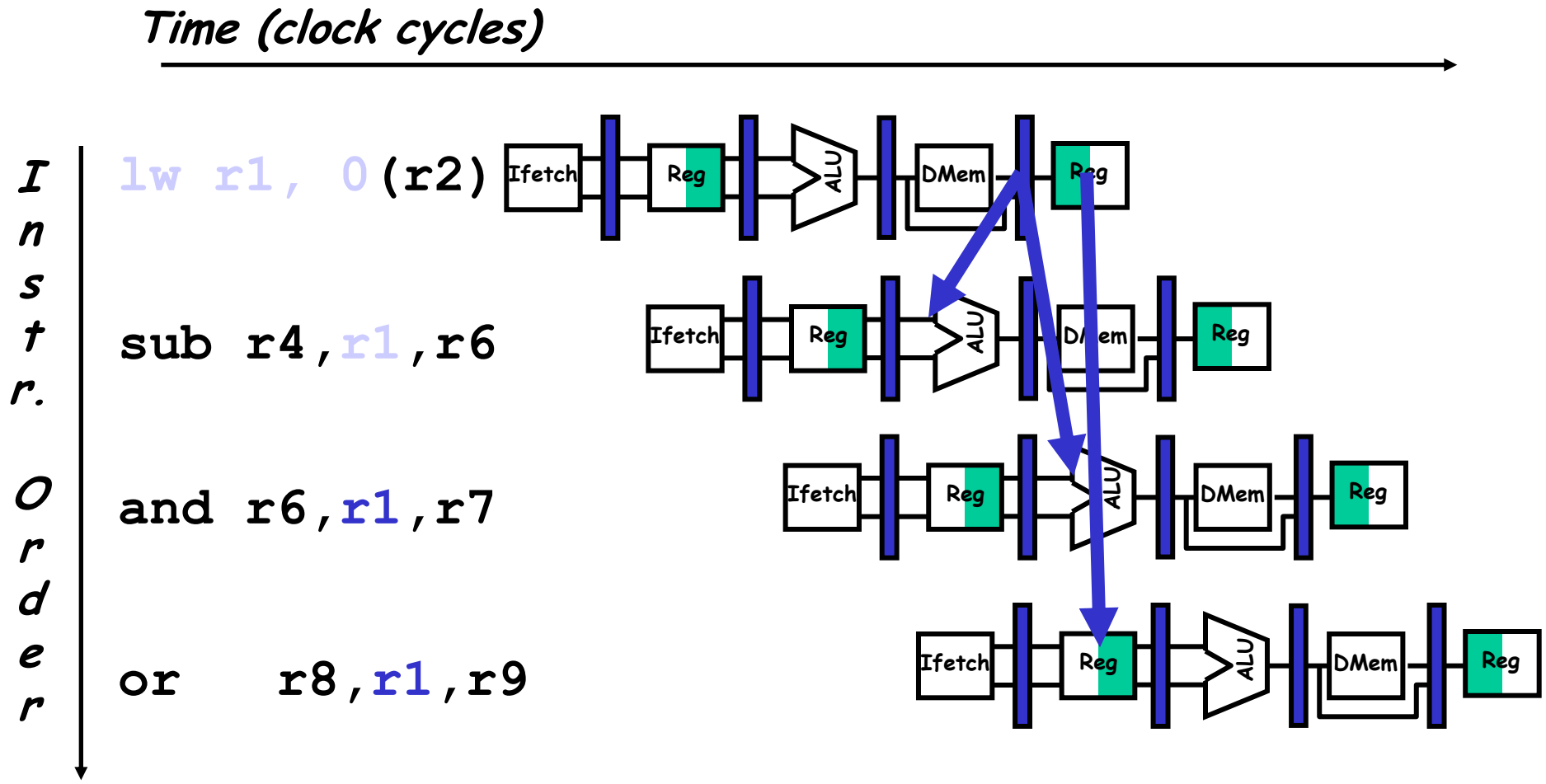
IC-UNICAMP



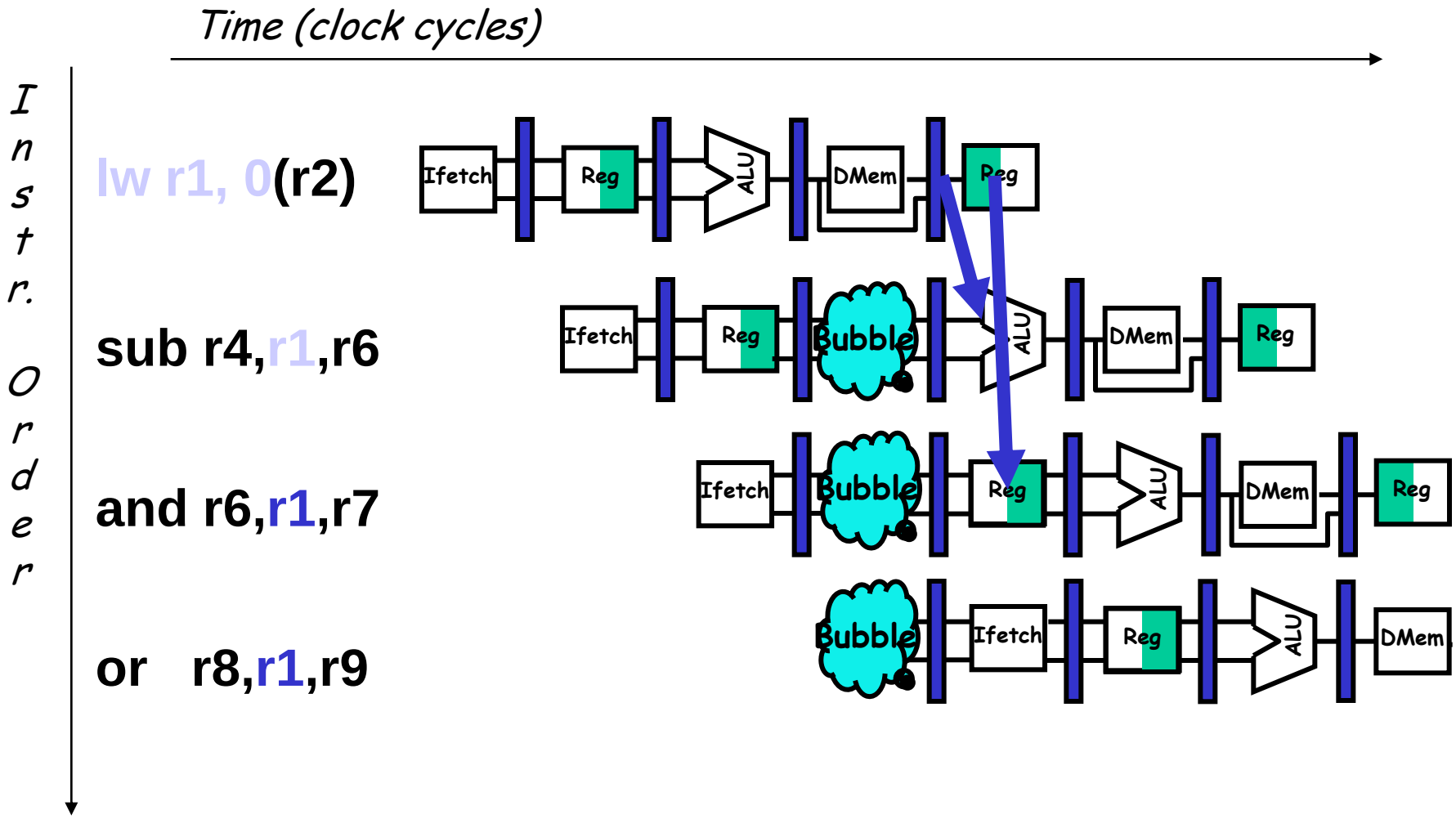
HW para Forwarding



Data Hazard com Forwarding



Data Hazard com Forwarding



Load Hazards - Software Scheduling

Código para (a, b, c, d ,e, f na memória).

a = b + c;

d = e – f;

Slow code:

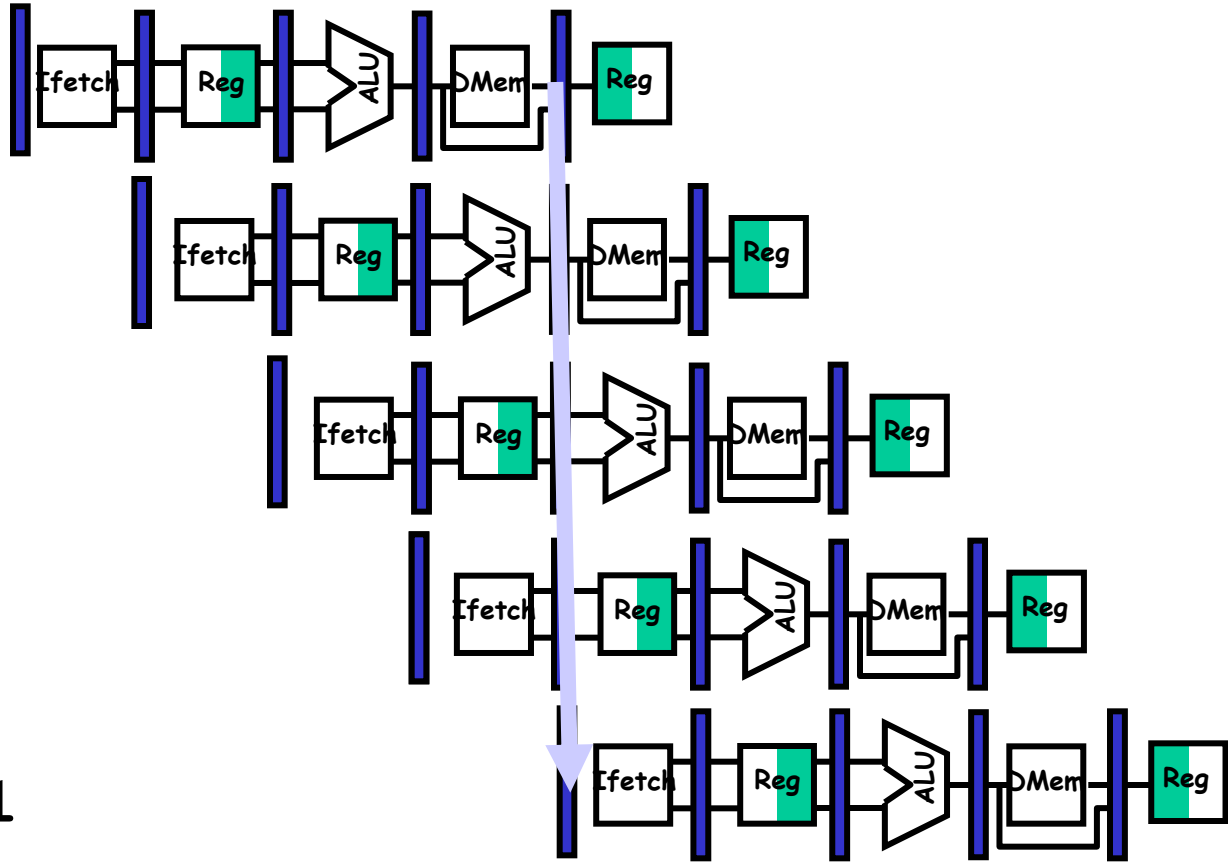
```
LW    Rb,b
LW    Rc,c
ADD   Ra,Rb,Rc
SW    a,Ra
LW    Re,e
LW    Rf,f
SUB   Rd,Re,Rf
SW    d,Rd
```

Fast code:

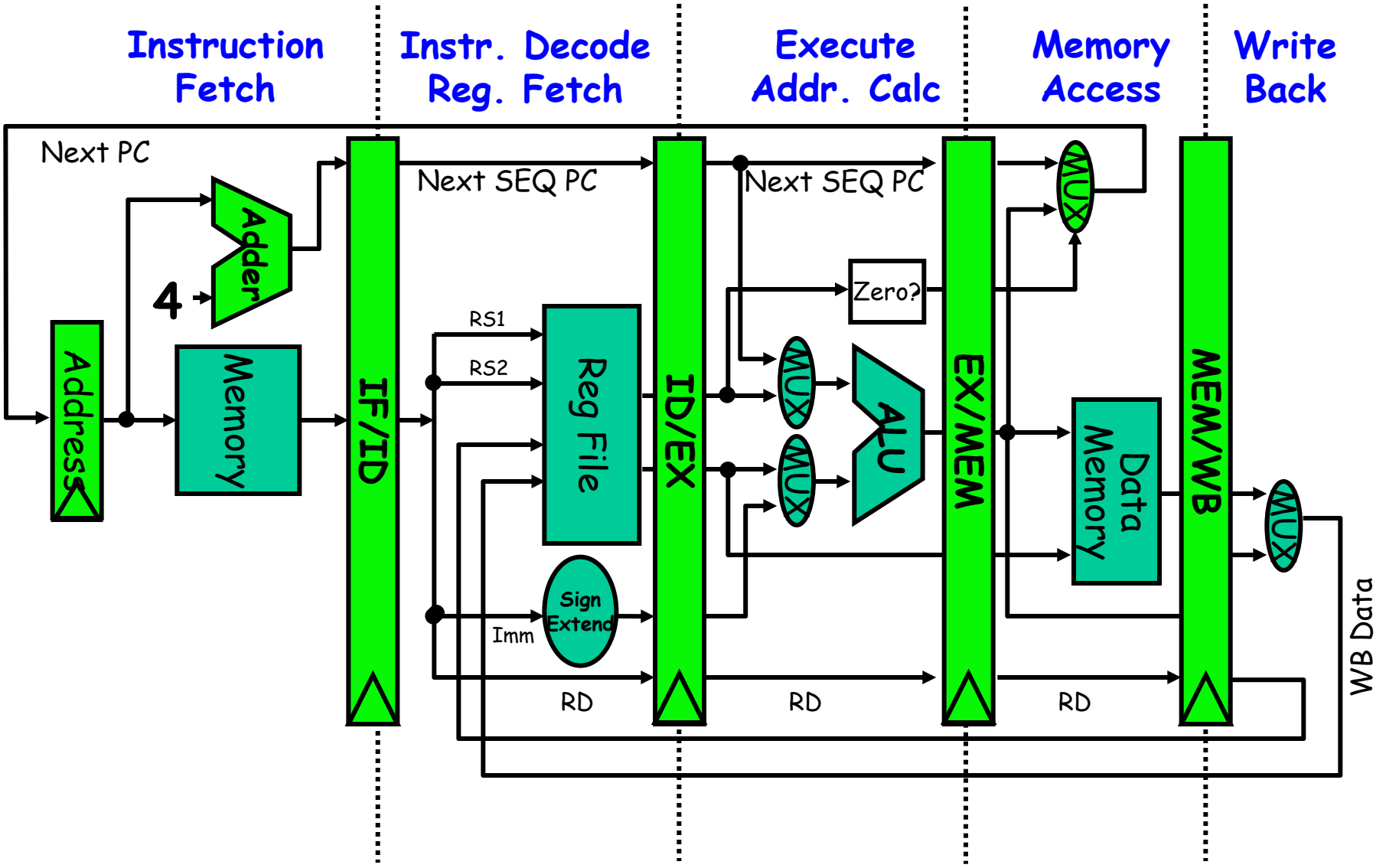
```
LW    Rb,b
LW    Rc,c
LW    Re,e
ADD   Ra,Rb,Rc
LW    Rf,f
SW    a,Ra
SUB   Rd,Re,Rf
SW    d,Rd
```

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
```



MIPS Datapath - 5 estágios



IC-UNICAMP

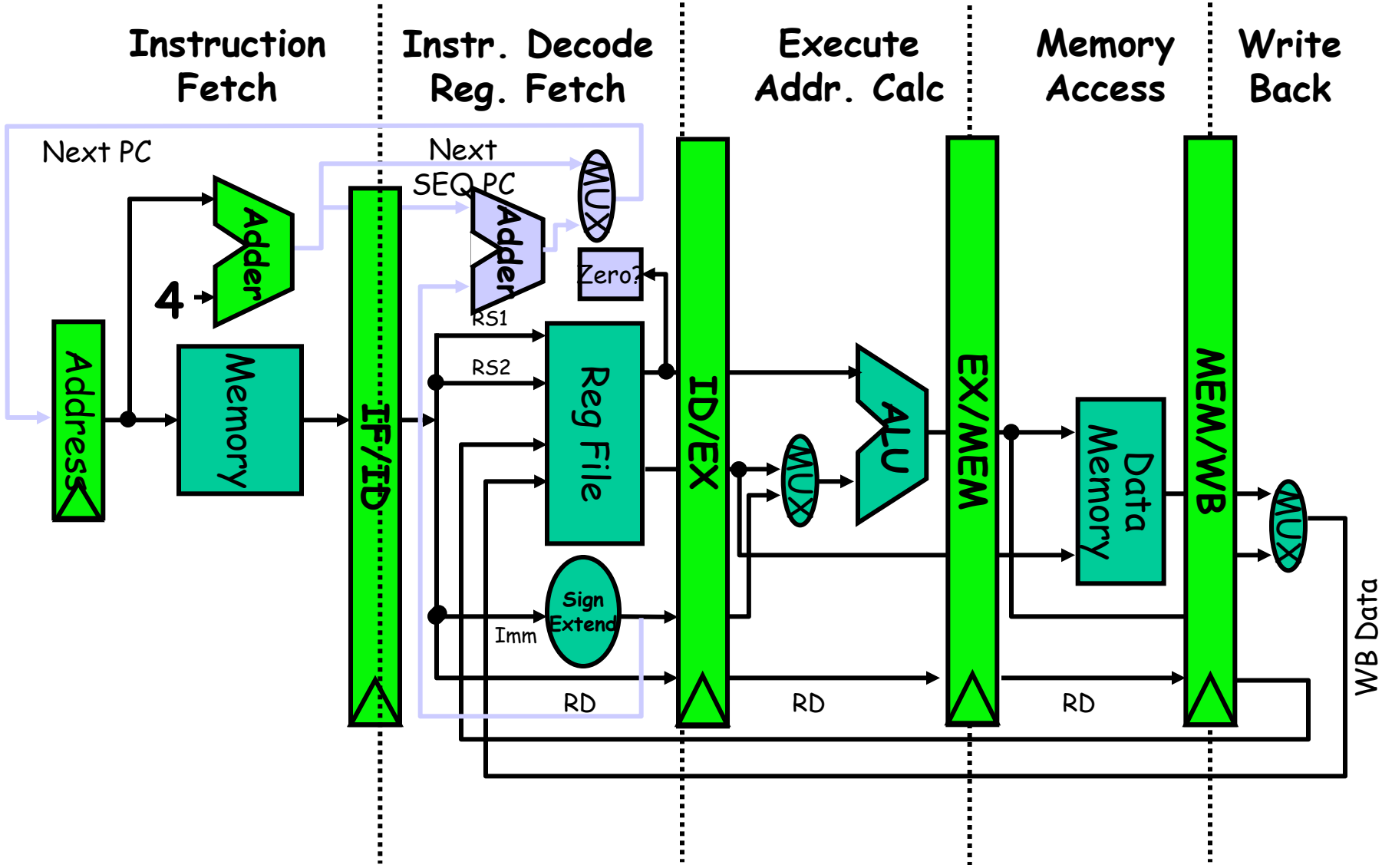
Exemplo: Impacto do Branch Stall

- 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 $\neq$ 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

Pipelined MIPS Datapath



IC-UNICAMP

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 tomado
- 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)

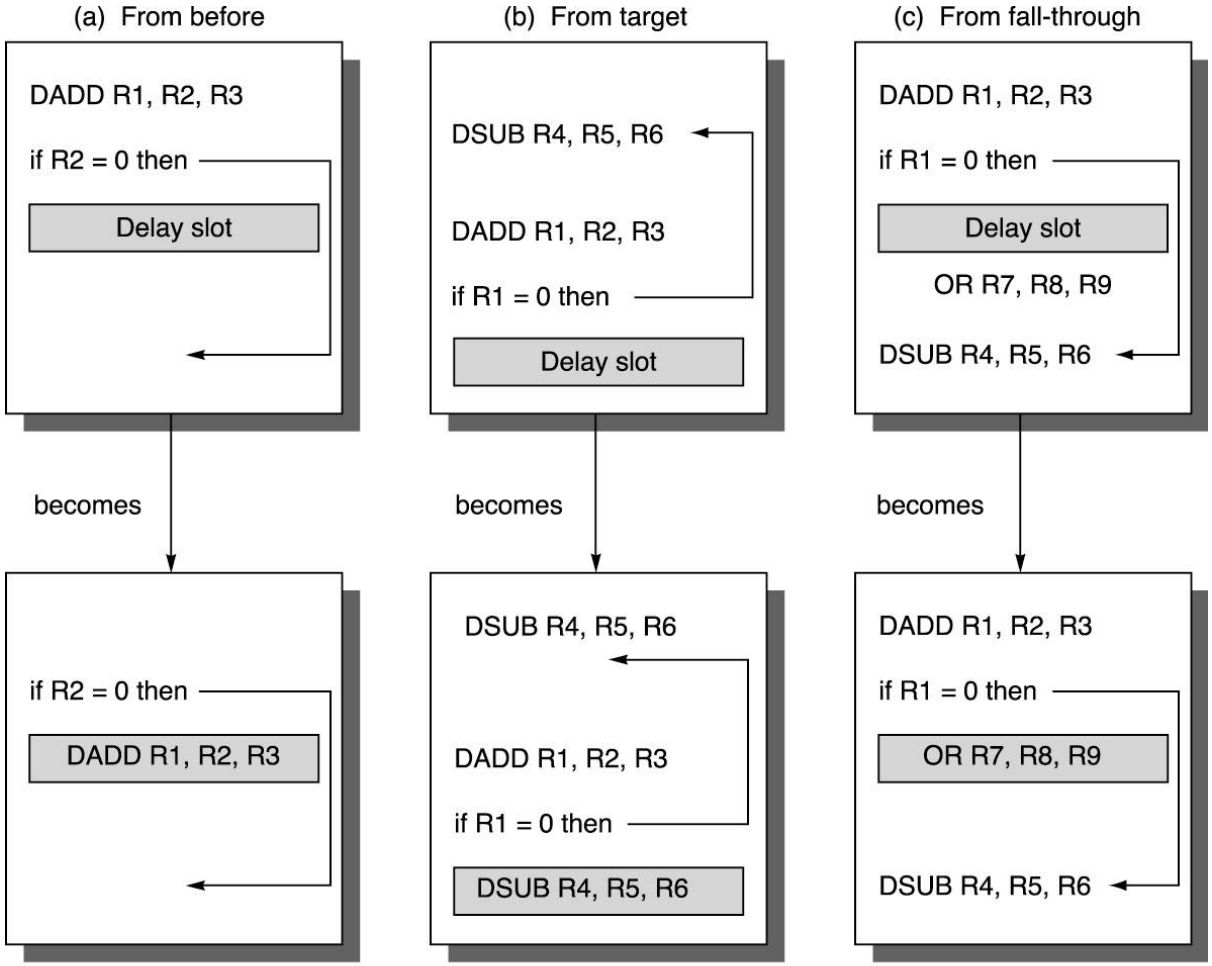
– 1 slot delay permite a decisão e o calculo do “branch target address” no pipeline de 5 estágios

– MIPS usa está solução

Delayed Branch

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

Delayed Branch



Apêndice C, Fig C14

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 preenchidos são utéis

Revisão: Desempenho

Qual o mais rápido?

Plane	DC to Paris	Speed	Passengers	Throughput (pmph)
Boeing 747	6.5 hours	610 mph	470	286,700
BAD/Sud Concorde	3 hours	1350 mph	132	178,200

- Time to run the task (ExTime)
  - Execution time, response time, latency
- Tasks per day, hour, week, sec, ns … (Desempenho)
  - Throughput, bandwidth

Definições

- Desempenho (performance) é em unidades por segundo
  - Quanto maior melhor

$$performance(x) = \frac{1}{execution\_time(x)}$$

" X é n vezes mais rápido que Y" significa que:

$$n = \frac{Performance(X)}{Performance(Y)} = \frac{Execution\_time(Y)}{Execution\_time(X)}$$

Aspectos sobre Desempenho de CPU (CPU Law)

$$\text{CPU time} = \frac{\text{Seconds}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Cycle}}$$

	Inst Count	CPI	Clock Rate
Program	X		
Compiler	X	(X)	
Inst. Set.	X	X	
Organization	X		X
Technology			X

Instruções por Ciclo (Throughput)

"Cycles per Instruction" médio

CPI = (CPU Time * Clock Rate) / Instruction Count
= Cycles / Instruction Count

$$\text{CPU time} = \text{Cycle Time} \times \sum_{j=1}^{n} \text{CPI}_j \times \text{I}_j$$

"Freqüência das Instruções"

$$\text{CPI} = \sum_{j=1}^{n} \text{CPI}_j \times \text{F}_j \qquad \text{where } \text{F}_j = \frac{\text{I}_j}{\text{Instruction Count}}$$

Exemplo: Calculando CPI

Base Machine (Reg / Reg)

Op	Freq	Cycles	CPI(i)	(% Time)
ALU	50%	1	.5	(33%)
Load	20%	2	.4	(27%)
Store	10%	2	.2	(13%)
Branch	20%	2	.4	(27%)
			1.5	

Mix típico de instruções em programas

IC-UNICAMP

Exemplo: Impacto de Branch Stall

- Assuma: CPI = 1.0 (ignorando branches stall por 3 ciclos)

- Se 30% são branch, Stall 3 ciclos

Op	Freq	Cycles	CPI(i)	(% Time)
Other	**70%**	**1**	**.7**	**(37%)**
Branch	**30%**	**4**	**1.2**	**(63%)**

- => novo CPI = 1.9, ou aproximadamente 2 vezes mais lento

IC-UNICAMP

Desempenho pipelines com stalls (C-12)

$$\text{Speedup from pipelining} = \frac{\text{Average instruction time unpipelined}}{\text{Average instruction time pipelined}}$$

$$= \frac{\text{CPI unpipelined} \times \text{Clock cycle unpipelined}}{\text{CPI pipelined} \times \text{Clock cycle pipelined}}$$

$$= \frac{\text{CPI unpipelined}}{\text{CPI pipelined}} \times \frac{\text{Clock cycle unpipelined}}{\text{Clock cycle pipelined}}$$

- Objetivo do Pipeline: diminuir CPI ou cycletime
- Primeiro caso: diminuir CPI (CPI ideal com pipeline = 1)

$$\text{CPI pipelined} = \text{Ideal CPI} + \text{Pipeline stall clock cycles per instruction}$$

$$= 1 + \text{Pipeline stall clock cycles per instruction}$$

- Assumindo overhead=0, pipeline balanceado, cycletimes iguais

$$\text{Speedup} = \frac{\text{CPI unpipelined}}{1 + \text{Pipeline stall cycles per instruction}}$$

- Caso especial (frequente), latência de todas instruções = estágios

$$\text{Speedup} = \frac{\text{Pipeline depth}}{1 + \text{Pipeline stall cycles per instruction}}$$

- Intuição OK: se stall = 0, speedup = pipeline depth

Desempenho pipelines com stalls (cont)

- Segundo caso: diminuir cycletime → CPI = 1 com ou sem pipeline

$$\text{Speedup from pipelining} = \frac{\text{CPI unpipelined}}{\text{CPI pipelined}} \times \frac{\text{Clock cycle unpipelined}}{\text{Clock cycle pipelined}}$$

$$= \frac{1}{1 + \text{Pipeline stall cycles per instruction}} \times \frac{\text{Clock cycle unpipelined}}{\text{Clock cycle pipelined}}$$

- Se pipeline balanceado e overhead = 0 → ganho no cycletime = pipeline depth

$$\text{Clock cycle pipelined} = \frac{\text{Clock cycle unpipelined}}{\text{Pipeline depth}}$$

$$\text{Pipeline depth} = \frac{\text{Clock cycle unpipelined}}{\text{Clock cycle pipelined}}$$

$$\text{Speedup from pipelining} = \frac{1}{1 + \text{Pipeline stall cycles per instruction}} \times \frac{\text{Clock cycle unpipelined}}{\text{Clock cycle pipelined}}$$

$$= \frac{1}{1 + \text{Pipeline stall cycles per instruction}} \times \text{Pipeline depth}$$

- Intuição OK: se stall = 0, speedup = pipeline depth

Exemplo 3: Branch Alternativas (C-25?)

$$\text{Pipeline speedup} = \frac{\text{Pipeline depth}}{1 + \text{Branch frequency} \times \text{Branch penalty}}$$

Scheduling scheme	*Branch penalty*	*CPI*	*speedup v. stall*
Stall pipeline	**3**	**1.42**	**1.0**
Predict taken	**1**	**1.14**	**1.26**
Predict not taken	**1**	**1.09**	**1.29**
Delayed branch	**0.5**	**1.07**	**1.31**

Exemplo 4: Dual-port vs. Single-port

- Máquina A: Dual ported memory (“Harvard Architecture”)
- Máquina B: Single ported memory, porém seu pipelined é 1.05 vezes mais rápido (clock rate)
- CPI Ideal = 1 para ambas
- Loads: 40% das instruções executadas

$SpeedUp_A$ = Pipeline Depth/(1 + 0) x ($clock_{unpipe}$/$clock_{pipe}$)
= Pipeline Depth

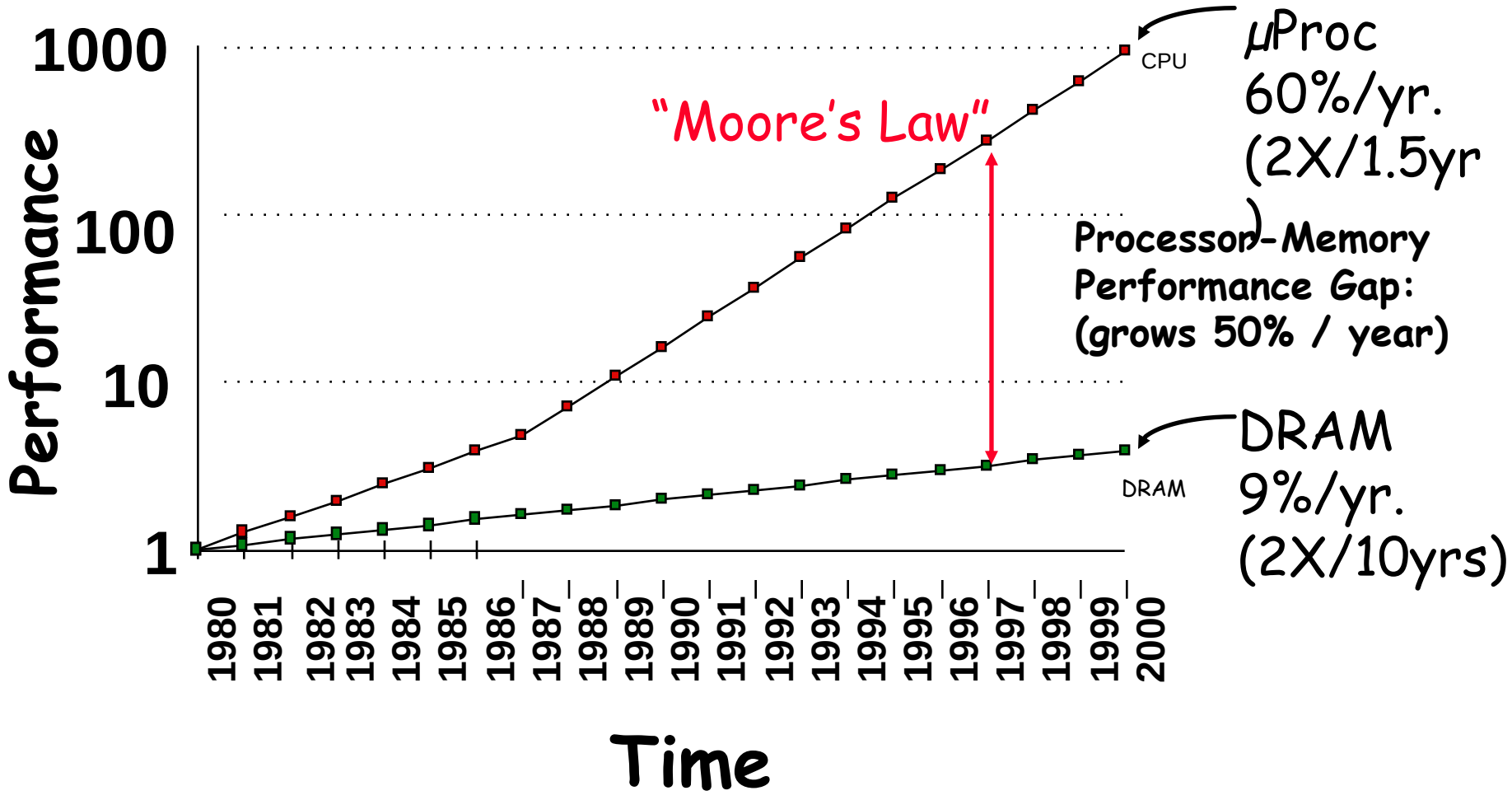
$SpeedUp_B$ = Pipeline Depth/(1 + 0.4 x 1) x ($clock_{unpipe}$/($clock_{unpipe}$ / 1.05)
= (Pipeline Depth/1.4) x 1.05
= 0.75 x Pipeline Depth

$SpeedUp_A$ / $SpeedUp_B$ = Pipeline Depth/(0.75 x Pipeline Depth) = 1.33

- Máquina A é 1.33 mais rápida que a B

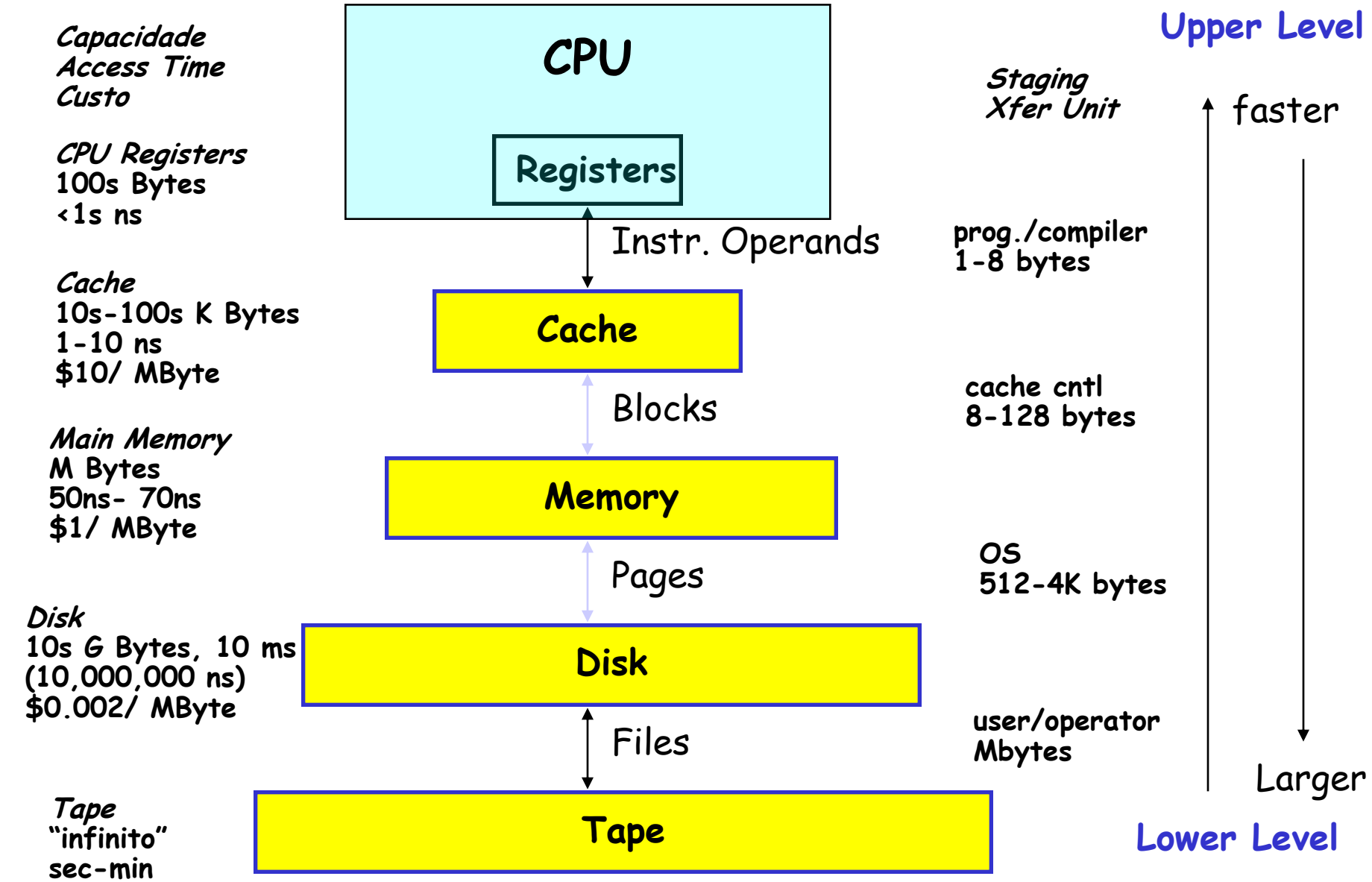
Revisão: Hierarquia de Memórias

Processor-DRAM Memory Gap (latency



Níveis em uma Hierarquia de Memórias

IC-UNICAMP

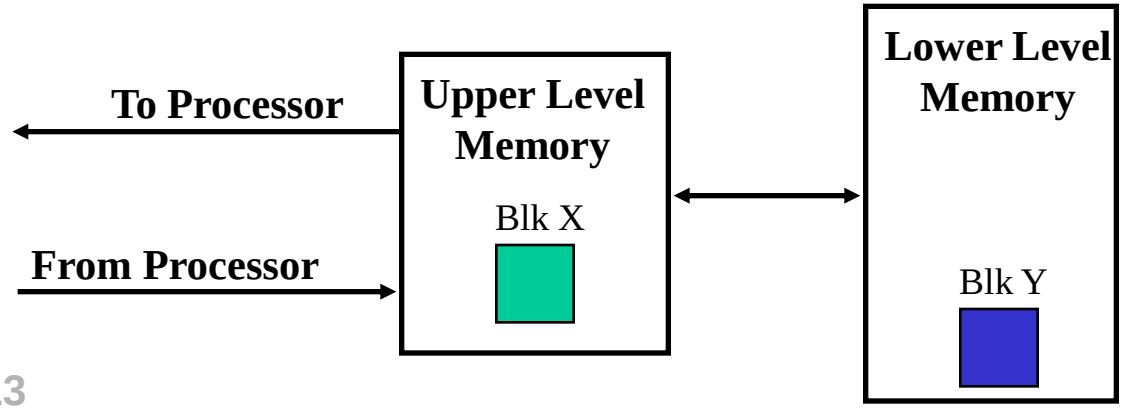


Princípio da Localidadde

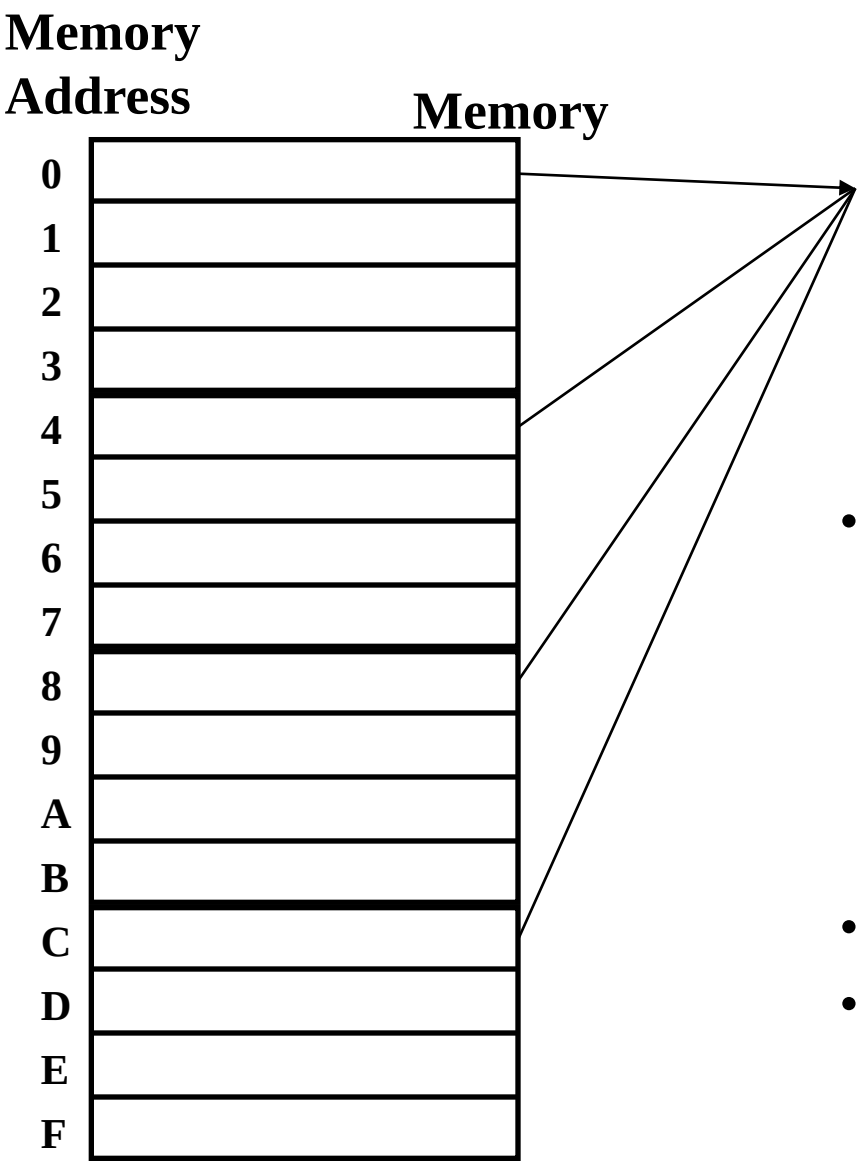
- Princípio da Localidade:
  - Programas acessam relativamente uma pequena porção do espaço de endereçamento em um dado instante de tempo.
- Dois tipos de Localidade:
  - Localidadde Temporal (Localidade no Tempo): Se um item é referenciado, ele tende a ser referenciado outra vez em um curto espaço de tempo (loops)
  - Localidade Espacial (Localidade no Espaço): Se um item é referenciado, itens próximos também tendem a serem referenciados em um curto espaço de tempo (acesso a array)

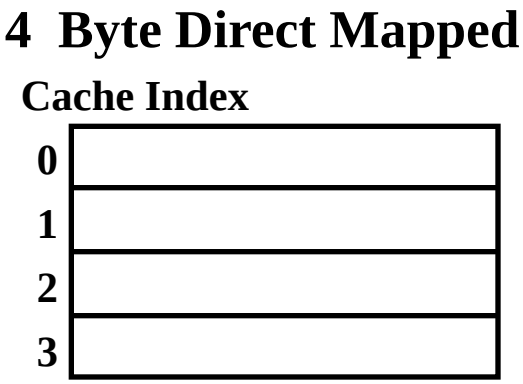
Hierarquia de Memórias: Terminologia

- Hit: o dado está no upper level (exemplo: Block X)
  - Hit Rate: taxa de hit no upper level no acesso à memória
  - Hit Time: Tempo para o acesso no upper level, consiste em:
    RAM access time + Time to determine hit/miss
- Miss: o dado precisa ser buscado em um bloco no lower level (Block Y)
  - Miss Rate = 1 - (Hit Rate)
  - Miss Penalty: Tempo para colocar um bloco no upper level + Tempo para disponibilizar o dado para o processador
- Hit Time << Miss Penalty (500 instruções no 21264!)



Cache: Direct Mapped

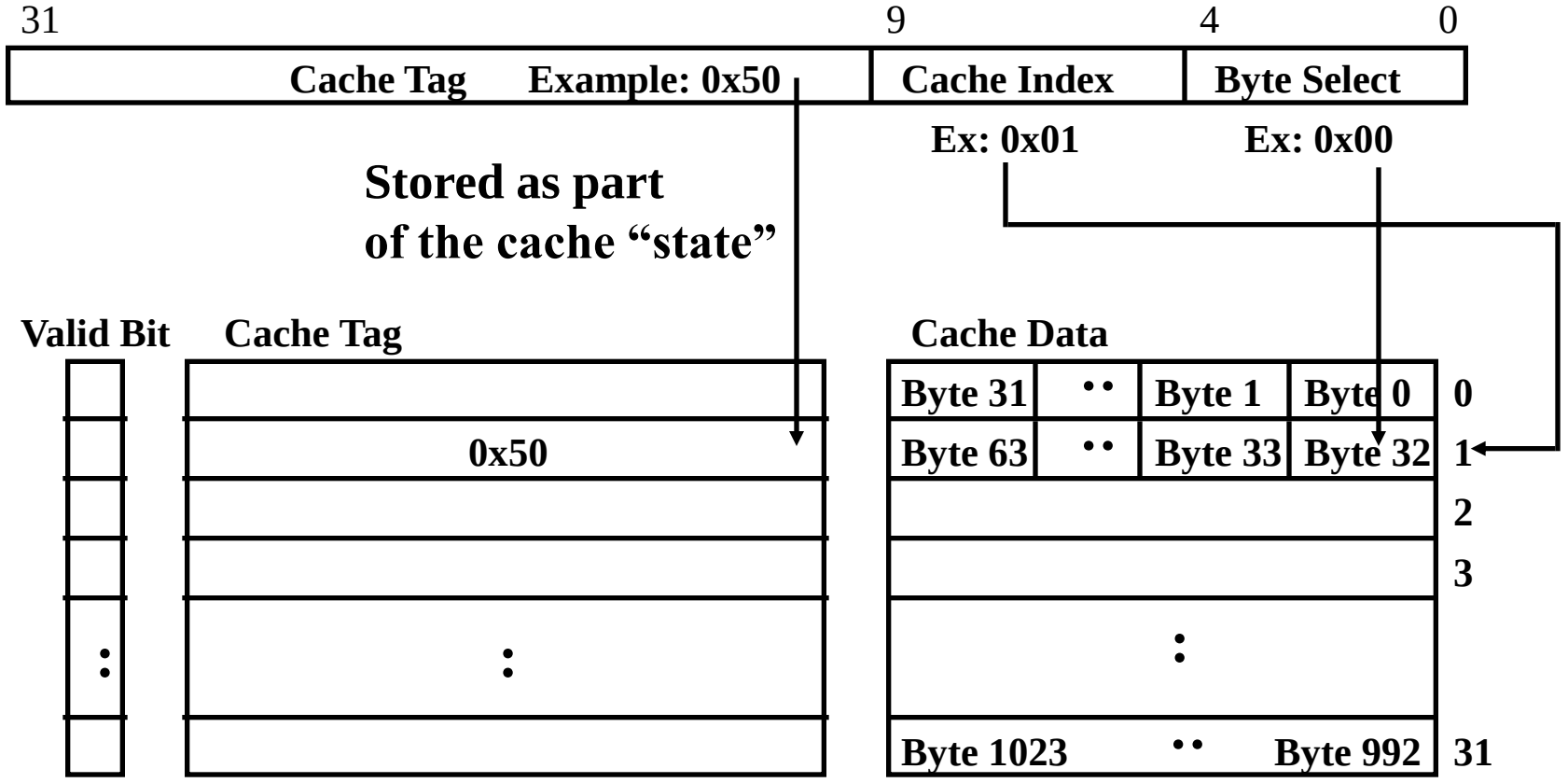




- Posição 0 pode ser ocupada por dados dos endereços em memória:
  - 0, 4, 8, ... etc.
  - Em geral: qq endereço cujos 2 LSBs são 0s
  - Address<1:0> => cache index
- Qual dado deve ser colocado na cache?
- Como definir o local, na cache, para cada dado?

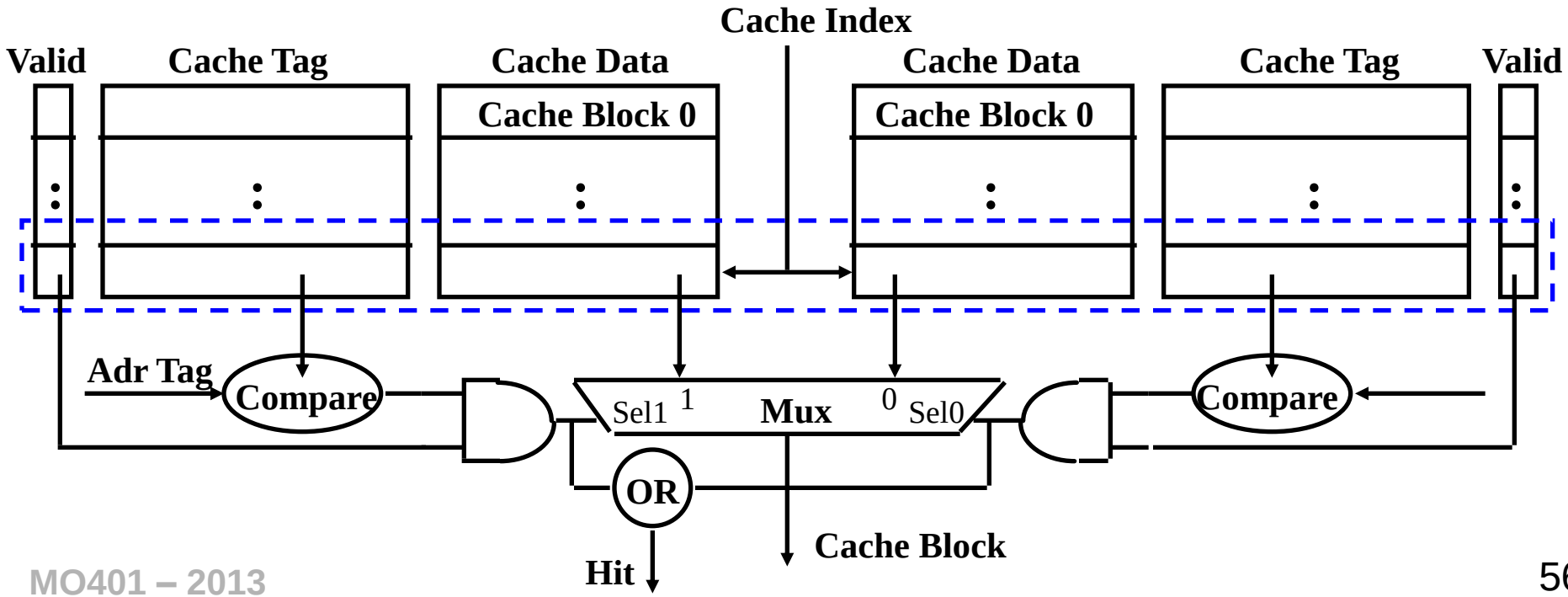
1 KB Direct Mapped Cache, 32B blocks

- Para uma cache de 2 ** N byte:
  - Os (32 - N) bits de mais alta ordem são “Cache Tag”
  - Os M bits de mais baixa ordem são “Byte Select” (Block Size = 2 ** M)



Two-way Set Associative Cache

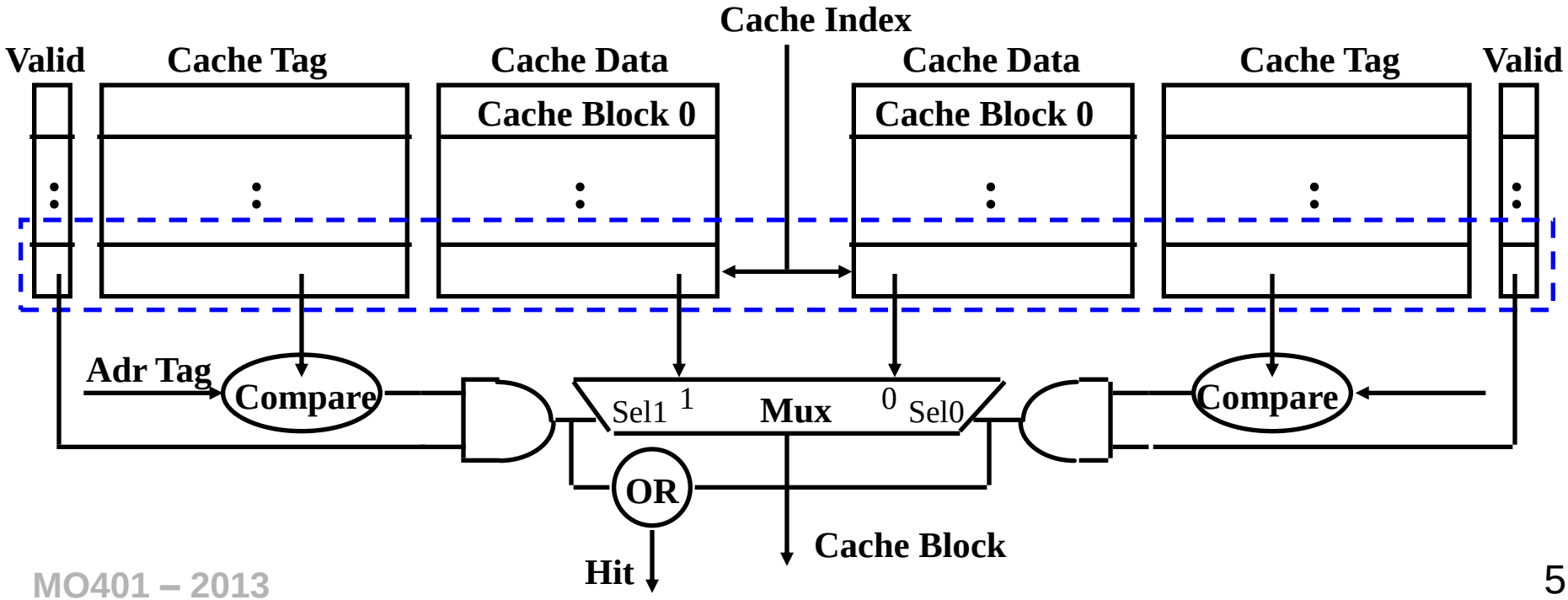
- N-way set associative: N entradas para cada Cache Index
  - N direct mapped caches opera em paralelo (N típico: 2 a 4)
- Examplo: Two-way set associative cache
  - Cache Index: seleciona um "set" na cache
  - As duas tags no set são comparadas em paralelo
  - O Dado é selecionado basedo no resultado da comparação das tag



Desvantagem de Set Associative Cache

- N-way Set Associative Cache v. Direct Mapped Cache:
  - N comparadores vs. 1
  - MUX extra, atrasa o acesso ao dado
  - Dado disponível APÓS Hit/Miss
- Direct mapped cache: Cache Block disponível ANTES do Hit/Miss:

  É possível assumir um hit e continuar. Se miss, Recover.



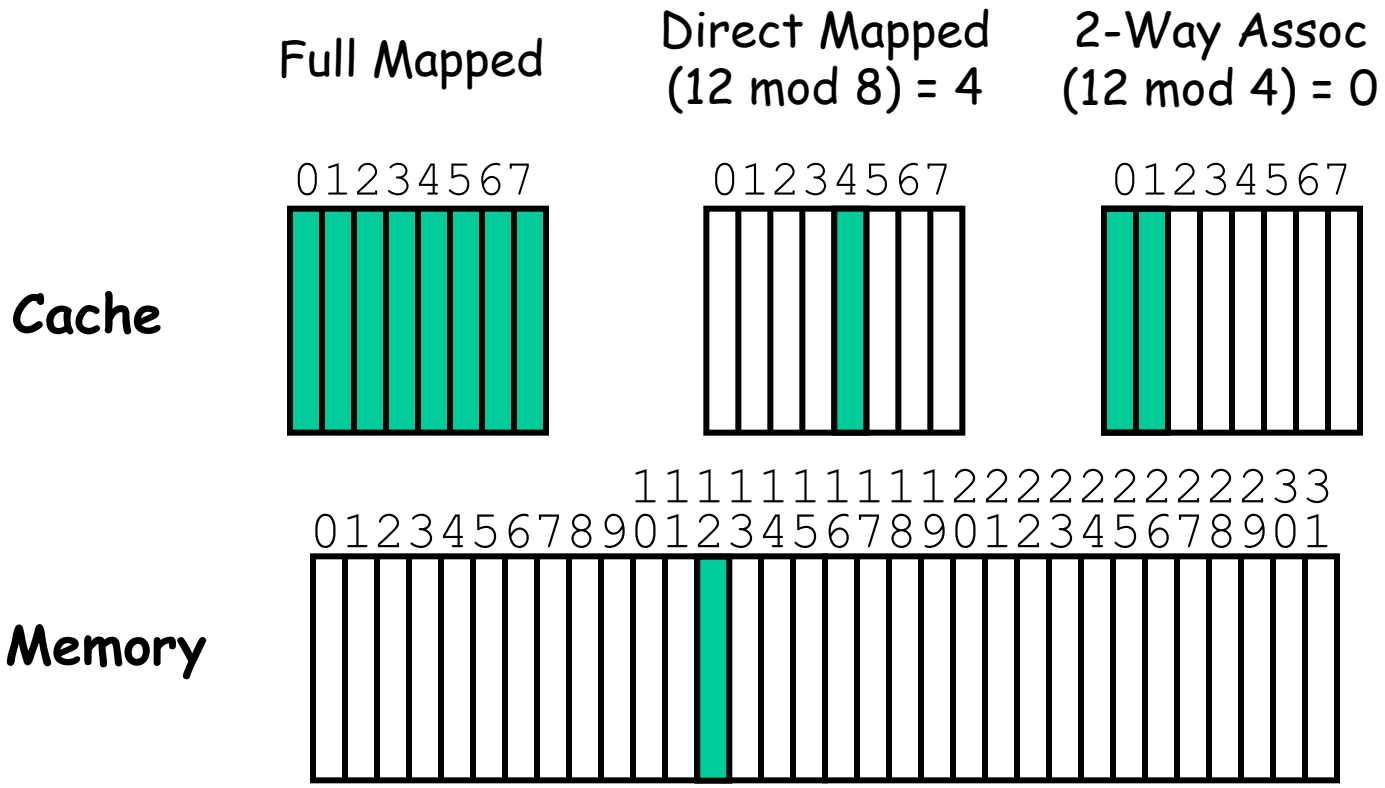
IC-UNICAMP

Hierarquia de Memória: 4 Questões

- Q1: Em que lugar colocar um bloco no upper level? → *(Block placement)*
- Q2: Como localizar o bloco se ele está no upper level? → *(Block identification)*
- Q3: Qual bloco deve ser trocado em um miss? → *(Block replacement)*
- Q4: O que ocorre em um write? → .*(Write strategy)*

Q1: Em que lugar colocar um bloco no upper level?

- Colocar o Bloco 12 em uma cache de 8 blocos :
  - Fully associative, direct mapped, 2-way set associative
  - S.A. Mapping = Block Number Modulo Number Sets



IC-UNICAMP

Q2: Como localizar o bloco se ele está no upper level?

- Tag em cada bloco
  - Não é necessário testar o index ou block offset

- O aumento da associatividade reduz o index e aumenta a tag

IC-UNICAMP

Q3: Qual bloco deve ser trocado em um miss?

- Fácil para Direct Mapped
- Set Associative or Fully Associative:
  - Random
  - LRU (Least Recently Used)
  - FIFO

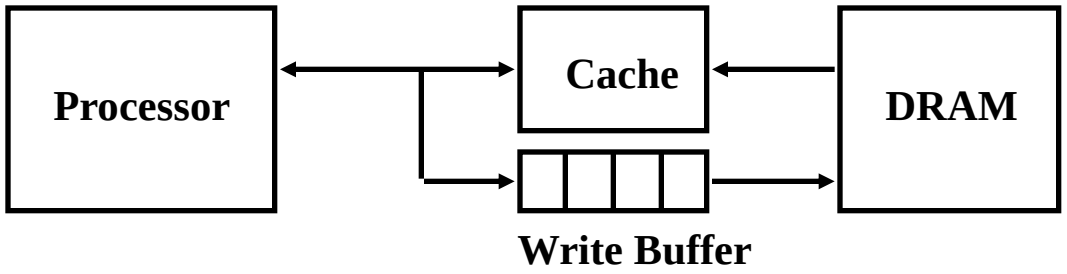
Assoc	2-way		4-way		8-way	
Size	LRU	Ran	LRU	Ran	LRU	Ran
16 KB	5.2%	5.7%	4.7%	5.3%	4.4%	5.0%
64 KB	1.9%	2.0%	1.5%	1.7%	1.4%	1.5%
256 KB	1.15%	1.17%	1.13%	1.13%	1.12%	1.12%

Taxa de misses na cache: blocos de 16 bytes na arquitetura Vax; acessos de usuários e do sistema operacional

Q4: O que ocorre em um write?

- *Write through* — A informação é escrita tanto no bloco da cache quanto no bloco do lower-level memory.
- *Write back* — A informação é escrita somente no bloco da cache. O bloco da cache modificado é escrito na memória principal somente quando ele é trocaddo.
  - block clean or dirty?
- Prós e Contras?
  - WT: read misses não pode resultar em writes
  - WB: não há repetição de writes na mesma posição
- WT, em geral, é combinado com write buffers, assim não há espera pelo lower level memory

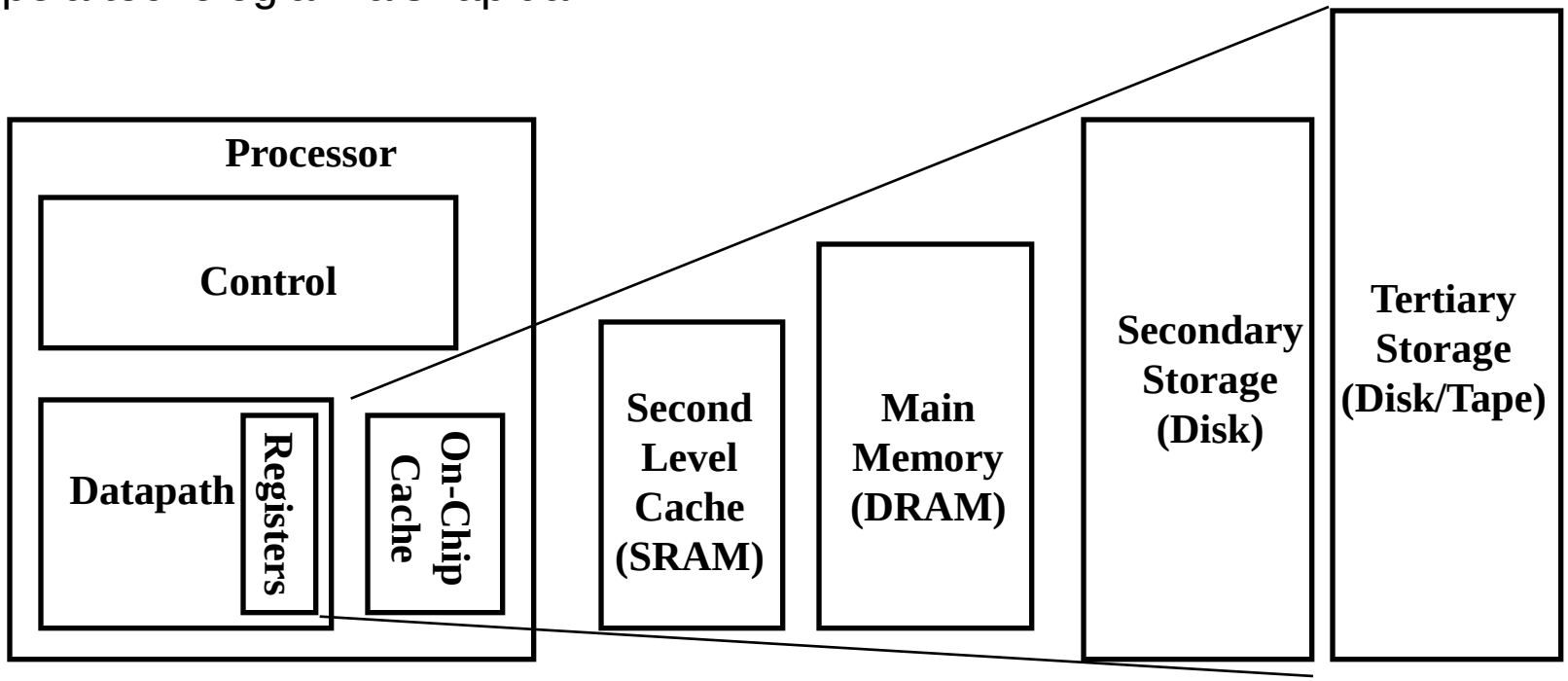
Write Buffer para Write Through



- Um Write Buffer colocado entre a Cache e a Memory
  - Processador: escreve o dado na cache e no write buffer
  - Memory controller: escreve o conteúdo do buffer na memória
- Write buffer é uma FIFO:
  - Número típico de entradas: 4
  - Trabalha bem se: freqüência de escrita (w.r.t. time) << 1 / DRAM write cycle
- Memory system é um pesadelo para o projetista :
  - freqüência de escrita (w.r.t. time) -> 1 / DRAM write cycle
  - Saturação do Write buffer

Hierarquia de Memória Moderna

- Tirando vantagens do príncipio da localidade:
  - Provê ao usuário o máximo de memória disponibilizada pela tecnologia mais barata.
  - Provê acesso na velocidade oferecida pela tecnologia mais rápida.



Speed (ns):	1s	10s	100s	10,000,000s (10s ms)	10,000,000,000s (10s sec)
Size (bytes):	100s	Ks	Ms	Gs	Ts

Resumo #1/3: Pipelining & Desempenho

- Sobreposição de tarefas; fácil se as tarefas são independentes
- Speed Up ≤ Pipeline Depth; Se CPI ideal for 1, então:

$$\text{Speedup} = \frac{\text{Pipeline depth}}{1 + \text{Pipeline stall CPI}} \times \frac{\text{Cycle Time}_{\text{unpipelined}}}{\text{Cycle Time}_{\text{pipelined}}}$$

- Hazards limita o desempenho nos computadores:
  - Estrutural: é necessário mais recursos de HW
  - Dados (RAW,WAR,WAW): forwarding, compiler scheduling
  - Controle: delayed branch, prediction
- **Tempo é a medida de desempenho: latência ou throughput**
- **CPI Law:**

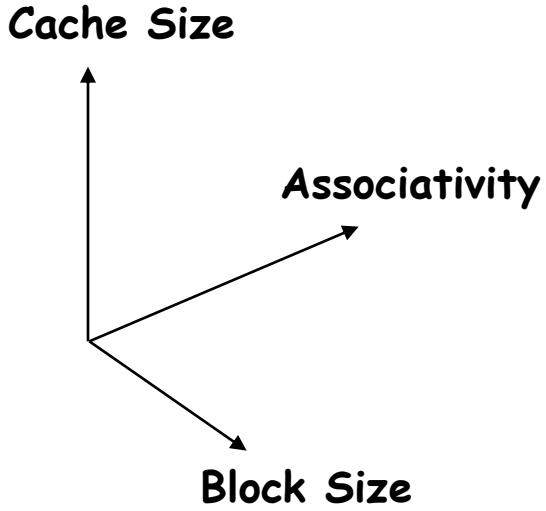
$$\text{CPU time} = \frac{\text{Seconds}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Cycle}}$$

Resumo #2/3: Caches

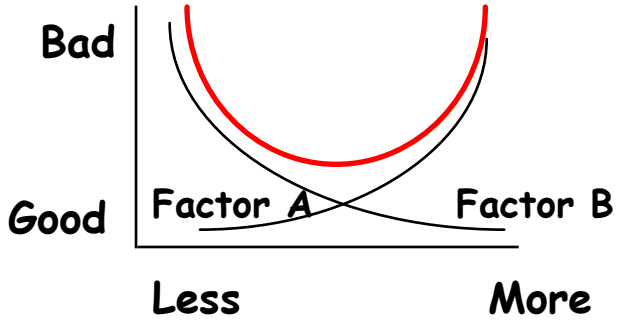
- Princípio da Localidade:
  - Programas acessam relativamente uma pequena porção do espaço de endereçamento em um dado instante de tempo.
    - Localidadde Temporal :Localidade no Tempo
    - Localidade Espacial : Localidade no Espaço
- Cache Misses: 3 categorias
  - Compulsory Misses
  - Capacity Misses
  - Conflict Misses
- Políticas de Escrita:
  - Write Through: (write buffer)
  - Write Back

Resumo #3/3: Cache Design

- Várias Dimensões interagindo
  - cache size
  - block size
  - associativity
  - replacement policy
  - write-through vs write-back



- Solução ótima é um compromiso
  - Depende da característica dos acessos
    - workload
    - I-cache, D-cache, TLB
  - Depende da razão tecnologia / custo



Revisão: ISA MIPS64

Conjunto de instruções: MIPS64

- Referência: Apêndice A (CAQA5)
- MIPS64 é usado em todo o curso como base para análise de todas as questões e problemas

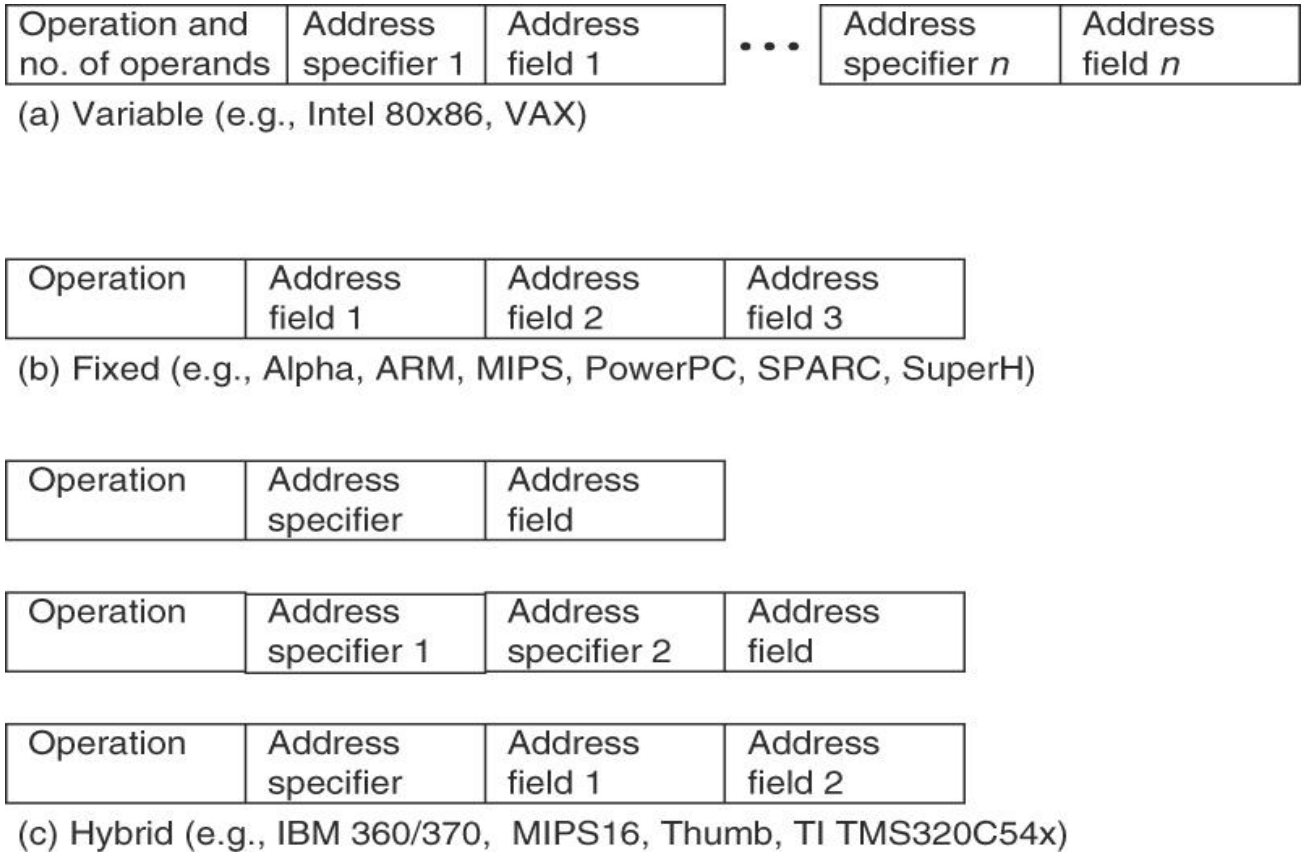


Figure A.18 Three basic variations in instruction encoding: variable length, fixed length, and hybrid. The variable format can support any number of operands, with each address specifier determining the addressing mode and the length of the specifier for that operand. It generally enables the smallest code representation, since unused fields need not be included. The fixed format always has the same number of operands, with the addressing modes (if options exist) specified as part of the opcode. It generally results in the largest code size. Although the fields tend not to vary in their location, they will be used for different purposes by different instructions. The hybrid approach has multiple formats specified by the opcode, adding one or two fields to specify the addressing mode and one or two fields to specify the operand address.

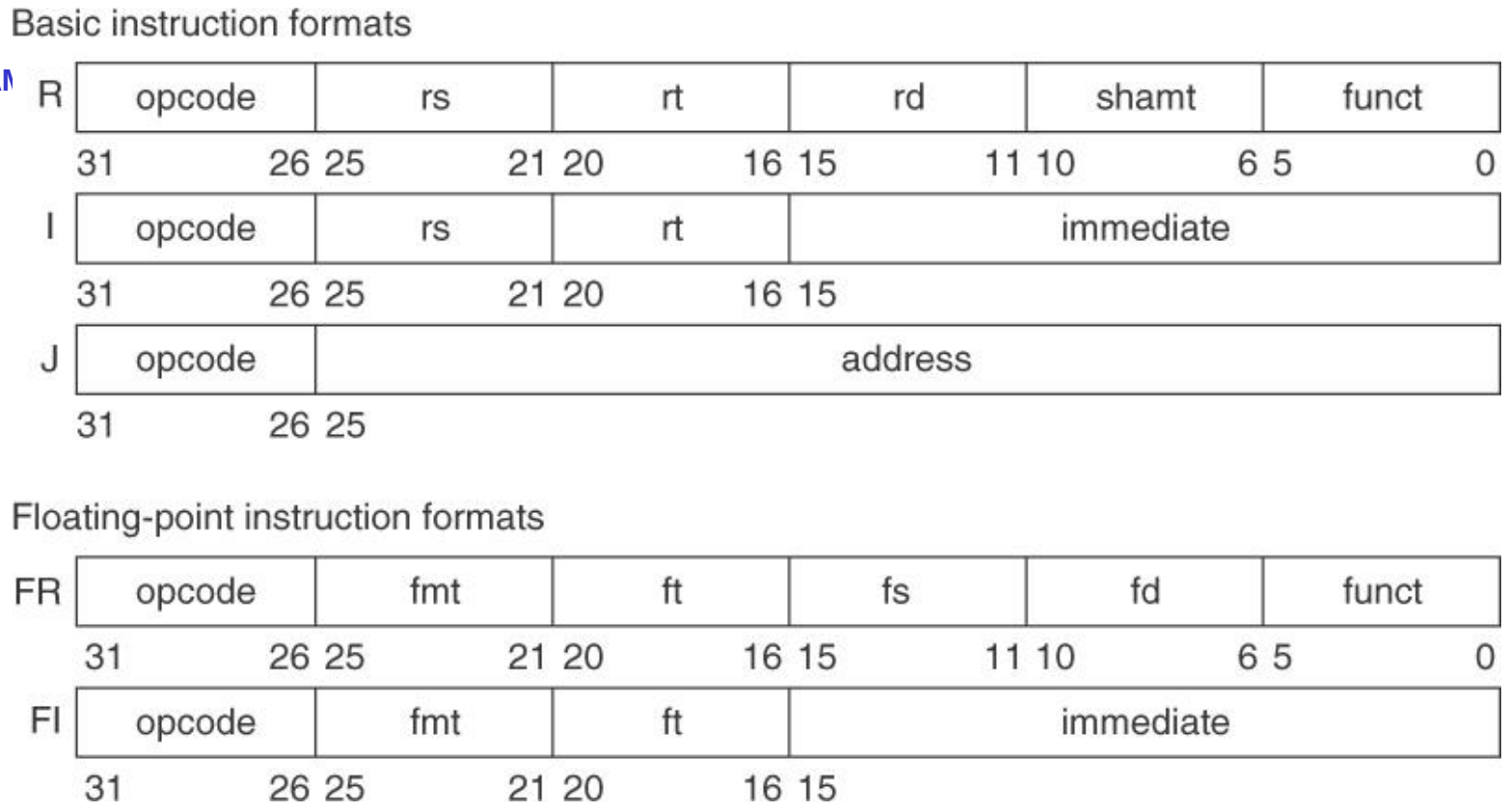


Figure 1.6 MIPS64 instruction set architecture formats. All instructions are 32 bits long. The R format is for integer register-to-register operations, such as DADDU, DSUBU, and so on. The I format is for data transfers, branches, and immediate instructions, such as LD, SD, BEQZ, and DADDIs. The J format is for jumps, the FR format for floating-point operations, and the FI format for floating-point branches.

Registradores

- Integer Registers
  - 32 registradores de 64 bits: GPR (general purpose registers) → R0, R1, ... , R31
  - R0 = 0 (sempre)
- Floating Point Registers
  - 32 registradores de 64 bits: FPR → F0, F1, ... , F31
  - permitem armazenar 32 números FP de precisão simples (32b) ou dupla (64b)
  - se FPR contém nº de precisão simples → metade não é usada
- Há instruções para FPR ←→ GPR e para usar 2 dados c/ precisão simples em um único FPR

Tipos de dados

- Bytes
- Meia palavra: 16 bits
- Palavra: 32 bits
- Palavra dupla: 64 bits
  - inteiros
  - FP precisão simples (32b) ou dupla (64b)
- Operações sobre dados
  - Palavra e palavra dupla
  - loads de bytes, meia palavras e palavras → MSB completados com zeros ou sign extend

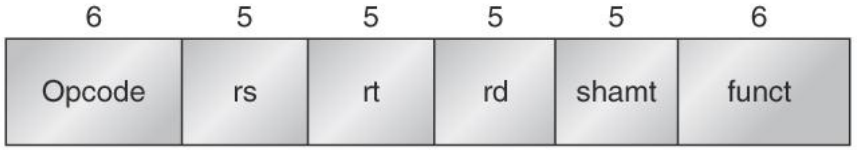
Formato das instruções

I-type instruction

Encodes: Loads and stores of bytes, half words, words, double words. All immediates (rt ← rs op immediate)

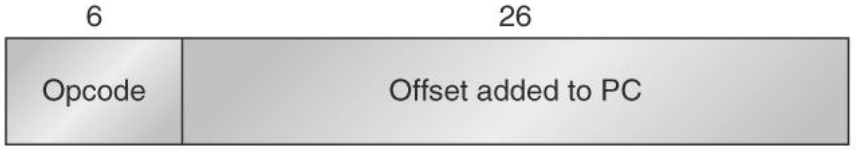
Conditional branch instructions (rs is register, rd unused)
Jump register, jump and link register
(rd = 0, rs = destination, immediate = 0)

R-type instruction



Register-register ALU operations: rd ← rs funct rt
Function encodes the data path operation: Add, Sub, . . .
Read/write special registers and moves

J-type instruction



Jump and jump and link
Trap and return from exception

Modos de endereçamento

- Imediato (16bits) e displacement
  - Endereço = conteúdo registrador + imediato
- Para endereçamento indireto registrador Ri
  - fazer imediato = 0
- Endereço de 64 bits, byte addressable
  - Mode bit→ SW pode selecionar big/little endian

Operações

- 4 classes:
  - loads e stores ; ALU; controle de fluxo; ponto flutuante
- Notação
  - bits dos registradores: 0 (MSB) → 63 (LSB)
  - Reg[R1] $\leftarrow_{64}$ Mem[30+Regs[R2]] : transferência de 64 bits da posição de memória 30+R2
  - $Regs[R4]_0$: bit 0 de R4
  - $Regs[R3]_{56..63}$: byte menos significativo de R3
  - 0^{48} : campo com 48 zeros
  - a ## b: a concatenado com b

Load e Store

Example instruction	Instruction name	Meaning
LD R1,30(R2)	Load double word	Regs[R1] $\leftarrow_{64}$ Mem[30+Regs[R2]]
LD R1,1000(R0)	Load double word	Regs[R1] $\leftarrow_{64}$ Mem[1000+0]
LW R1,60(R2)	Load word	Regs[R1] $\leftarrow_{64}$ (Mem[60+Regs[R2]]$_0$)^{32} ## Mem[60+Regs[R2]]
LB R1,40(R3)	Load byte	Regs[R1] $\leftarrow_{64}$ (Mem[40+Regs[R3]]$_0$)^{56} ## Mem[40+Regs[R3]]
LBU R1,40(R3)	Load byte unsigned	Regs[R1] $\leftarrow_{64}$ 0^{56} ## Mem[40+Regs[R3]]
LH R1,40(R3)	Load half word	Regs[R1] $\leftarrow_{64}$ (Mem[40+Regs[R3]]$_0$)^{48} ## Mem[40+Regs[R3]] ## Mem[41+Regs[R3]]
L.S F0,50(R3)	Load FP single	Regs[F0] $\leftarrow_{64}$ Mem[50+Regs[R3]] ## 0^{32}
L.D F0,50(R2)	Load FP double	Regs[F0] $\leftarrow_{64}$ Mem[50+Regs[R2]]
SD R3,500(R4)	Store double word	Mem[500+Regs[R4]] $\leftarrow_{64}$ Regs[R3]
SW R3,500(R4)	Store word	Mem[500+Regs[R4]] $\leftarrow_{32}$ Regs[R3]$_{32..63}$
S.S F0,40(R3)	Store FP single	Mem[40+Regs[R3]] $\leftarrow_{32}$ Regs[F0]$_{0..31}$
S.D F0,40(R3)	Store FP double	Mem[40+Regs[R3]] $\leftarrow_{64}$ Regs[F0]
SH R3,502(R2)	Store half	Mem[502+Regs[R2]] $\leftarrow_{16}$ Regs[R3]$_{48..63}$
SB R2,41(R3)	Store byte	Mem[41+Regs[R3]] $\leftarrow_{8}$ Regs[R2]$_{56..63}$

Figure A.23 The load and store instructions in MIPS. All use a single addressing mode and require that the memory value be aligned. Of course, both loads and stores are available for all the data types shown.

ALU

Example instruction	Instruction name	Meaning
DADDU R1,R2,R3	Add unsigned	Regs[R1] ← Regs[R2]+Regs[R3]
DADDIU R1,R2,#3	Add immediate unsigned	Regs[R1] ← Regs[R2]+3
LUI R1,#42	Load upper immediate	Regs[R1] ← 0^{32}##42##0^{16}
DSLL R1,R2,#5	Shift left logical	Regs[R1] ← Regs[R2]<<5
SLT R1,R2,R3	Set less than	if (Regs[R2]<Regs[R3]) Regs[R1] ← 1 else Regs[R1]←0

Figure A.24 Examples of arithmetic/logical instructions on MIPS, both with and without immediates.

Controle de fluxo

IC

Example instruction		Instruction name	Meaning
J	name	Jump	$PC_{36..63} \leftarrow$ name
JAL	name	Jump and link	Regs[R31] $\leftarrow$ PC+8; $PC_{36..63} \leftarrow$ name; $((PC + 4)-2^{27}) \leq$ name $< ((PC+4)+2^{27})$
JALR	R2	Jump and link register	Regs[R31] $\leftarrow$ PC+8; PC $\leftarrow$ Regs[R2]
JR	R3	Jump register	PC $\leftarrow$ Regs[R3]
BEQZ	R4,name	Branch equal zero	if (Regs[R4]==0) PC $\leftarrow$ name; $((PC+4)-2^{17}) \leq$ name $< ((PC+4)+2^{17})$
BNE	R3,R4,name	Branch not equal zero	if (Regs[R3] != Regs[R4]) PC $\leftarrow$ name; $((PC+4)-2^{17}) \leq$ name $< ((PC+4)+2^{17})$
MOVZ	R1,R2,R3	Conditional move if zero	if (Regs[R3]==0) Regs[R1] $\leftarrow$ Regs[R2]

Figure A.25 Typical control flow instructions in MIPS. All control instructions, except jumps to an address in a register, are PC-relative. Note that the branch distances are longer than the address field would suggest; since MIPS instructions are all 32 bits long, the byte branch address is multiplied by 4 to get a longer distance.

Ponto flutuante

Floating point	*FP operations on DP and SP formats*
ADD.D,ADD.S,ADD.PS	Add DP, SP numbers, and pairs of SP numbers
SUB.D,SUB.S,SUB.PS	Subtract DP, SP numbers, and pairs of SP numbers
MUL.D,MUL.S,MUL.PS	Multiply DP, SP floating point, and pairs of SP numbers
MADD.D,MADD.S,MADD.PS	Multiply-add DP, SP numbers, and pairs of SP numbers
DIV.D,DIV.S,DIV.PS	Divide DP, SP floating point, and pairs of SP numbers
CVT._._	Convert instructions: CVT.x.y converts from type x to type y, where x and y are L (64-bit integer), W (32-bit integer), D (DP), or S (SP). Both operands are FPRs.
C.__.D,C.__.S	DP and SP compares: "__" = LT,GT,LE,GE,EQ,NE; sets bit in FP status register

Instruction type/opcode	Instruction meaning
Data transfers	*Move data between registers and memory, or between the integer and FP or special registers; only memory address mode is 16-bit displacement + contents of a GPR*
LB,LBU,SB	Load byte, load byte unsigned, store byte (to/from integer registers)
LH,LHU,SH	Load half word, load half word unsigned, store half word (to/from integer registers)
LW,LWU,SW	Load word, load word unsigned, store word (to/from integer registers)
LD,SD	Load double word, store double word (to/from integer registers)
L.S,L.D,S.S,S.D	Load SP float, load DP float, store SP float, store DP float
MFC0,MTC0	Copy from/to GPR to/from a special register
MOV.S,MOV.D	Copy one SP or DP FP register to another FP register
MFC1,MTC1	Copy 32 bits to/from FP registers from/to integer registers
Arithmetic/logical	*Operations on integer or logical data in GPRs; signed arithmetic trap on overflow*
DADD,DADDI,DADDU,DADDIU	Add, add immediate (all immediates are 16 bits); signed and unsigned
DSUB,DSUBU	Subtract; signed and unsigned
DMUL,DMULU,DDIV, DDIVU,MADD	Multiply and divide, signed and unsigned; multiply-add; all operations take and yield 64-bit values
AND,ANDI	And, and immediate
OR,ORI,XOR,XORI	Or, or immediate, exclusive or, exclusive or immediate
LUI	Load upper immediate; loads bits 32 to 47 of register with immediate, then sign-extends
DSLL,DSRL,DSRA,DSLLV, DSRLV,DSRAV	Shifts: both immediate (DS__) and variable form (DS__V); shifts are shift left logical, right logical, right arithmetic
SLT,SLTI,SLTU,SLTIU	Set less than, set less than immediate; signed and unsigned
Control	*Conditional branches and jumps; PC-relative or through register*
BEQZ,BNEZ	Branch GPRs equal/not equal to zero; 16-bit offset from PC + 4
BEQ,BNE	Branch GPR equal/not equal; 16-bit offset from PC + 4
BC1T,BC1F	Test comparison bit in the FP status register and branch; 16-bit offset from PC + 4
MOVN,MOVZ	Copy GPR to another GPR if third GPR is negative, zero
J,JR	Jumps: 26-bit offset from PC + 4 (J) or target in register (JR)
JAL,JALR	Jump and link: save PC + 4 in R31, target is PC-relative (JAL) or a register (JALR)
TRAP	Transfer to operating system at a vectored address
ERET	Return to user code from an exception; restore user mode
Floating point	*FP operations on DP and SP formats*
ADD.D,ADD.S,ADD.PS	Add DP, SP numbers, and pairs of SP numbers
SUB.D,SUB.S,SUB.PS	Subtract DP, SP numbers, and pairs of SP numbers
MUL.D,MUL.S,MUL.PS	Multiply DP, SP floating point, and pairs of SP numbers
MADD.D,MADD.S,MADD.PS	Multiply-add DP, SP numbers, and pairs of SP numbers
DIV.D,DIV.S,DIV.PS	Divide DP, SP floating point, and pairs of SP numbers
CVT._._	Convert instructions: CVT.x.y converts from type x to type y, where x and y are L (64-bit integer), W (32-bit integer), D (DP), or S (SP). Both operands are FPRs.
C.__.D,C.__.S	DP and SP compares: "__" = LT,GT,LE,GE,EQ,NE; sets bit in FP status register

ISA de outros computadores

- Ver apêndice K