

# Arquitetura e Organização de Computadores

## Aula-13: Conjunto Reduzido de Instruções

Eliseu César Miguel

Livro: Arquitetura e Organização de Computadores  
William Stallings 8<sup>a</sup> Edição  
Universidade Federal de Alfenas

February 4, 2021



# Organização da Aula

- 1 Introdução
- 2 Fundamentos RISC
- 3 Pipeline de Instruções
- 4 Obstáculos do Pipeline
- 5 RISC & CISC

# Organização da Aula

- 1 Introdução
- 2 Fundamentos RISC
- 3 Pipeline de Instruções
- 4 Obstáculos do Pipeline
- 5 RISC & CISC

# Organização da Aula

- 1 Introdução
- 2 Fundamentos RISC
- 3 Pipeline de Instruções
- 4 Obstáculos do Pipeline
- 5 RISC & CISC

# Organização da Aula

- 1 Introdução
- 2 Fundamentos RISC
- 3 Pipeline de Instruções
- 4 Obstáculos do Pipeline
- 5 RISC & CISC

# Organização da Aula

- 1 Introdução
- 2 Fundamentos RISC
- 3 Pipeline de Instruções
- 4 Obstáculos do Pipeline
- 5 RISC & CISC

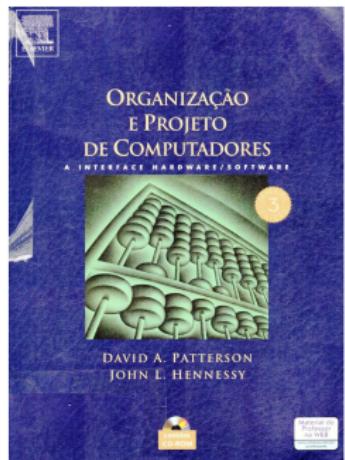
# Bibliografia Básica



Livro Texto



Complementar



Complementar

Além da bibliografia básica, visite o programa de ensino para ver outras bibliografias. Também, durante o curso, várias bibliografias em sítios da Internet serão apresentadas.

# Introdução

## Pontos importantes:

- Estudos do comportamento de execução dos programas de linguagem de alto nível forneceram um guia para projetar um novo tipo de arquitetura de processador: computador com conjunto reduzido de instruções (RISC — *reduced instructions set computer*). As instruções de atribuição predominam, sugerindo que os movimentos simples de dados deveriam ser otimizados. Há também muitas instruções de controle, o que sugere que o mecanismo de controle necessita ser otimizado para permitir *pipeline* eficiente. Estudos sobre padrões de referência de operando sugerem que deveria ser possível melhorar o desempenho guardando um número moderado de operandos nos registradores.
- Eses estudos motivaram as principais características das máquinas RISC:
  - ▶ um conjunto de instruções limitado com um formato fixo
  - ▶ número grande de registradores ou um compilador que otimize o uso de registradores
  - ▶ uma ênfase na otimização do pipeline de instruções
- O conjunto de instruções simples de uma máquina RISC leva por si só a um *pipeline* eficiente porque há menos operações por instrução e elas são mais previsíveis. Uma arquitetura com conjunto de instruções RISC também leva por si só à técnica de desvio atrasado, na qual instruções de desvio são rearranjadas com outras instruções para melhorar a eficiência do *pipeline*.

# Introdução

Alguns dos maiores avanços desde o nascimento do computador:

- **Conceito de família:** introduzido pela IBM com seu System/360 em 1964, seguido logo depois por DEC com seu PDP-8. As diferenças em preço e desempenho se devem às diferentes implementações da mesma arquitetura.
- **Unidade de controle microprogramada:** sugerida por Wilkes em 1951 e introduzida pela IBM na linha S/360 em 1964. A microprogramação facilita a tarefa de projetar e implementar a unidade de controle e fornece suporte para conceito de família.
- **Memória cache:** introduzida pela primeira vez comercialmente no S/360 Model 85 da IBM em 1968. A inserção deste elemento na hierarquia de memória melhora o desempenho consideravelmente.
- **Pipeline:** um meio para introduzir paralelismo na natureza essencialmente sequencial de um programa de instruções de máquina. Exemplos são pipeline de instruções e processamento vetorial. múltiplos processadores : esta categoria cobre um número de organizações e objetivos diferentes.
- **RISC** Arquitetura de computadores com conjunto de instruções reduzido.

# Introdução (Visto na Aula-12)

## Conceito: *Pipeline* de instruções

*Pipelining* é uma técnica de implementação na qual várias instruções são sobrepostas na execução. Esta técnica tira proveito do paralelismo que existe entre as ações necessárias para executar uma instrução.

## Conceitos necessários para o estudo de *pipeline*

- Fundamentos RISC;
- Arquitetura: Von Newmann & Arquitetura Harvard.

# Fundamentos RISC

## Elementos envolvidos nos na abordagem RISC:

- Um grande número de registradores de propósito geral e/ou o uso de tecnologia de compiladores para otimizar uso de registradores.  
(Arquitetura Reg/Reg)
- Um conjunto de instruções simples e limitado.  
(São implementadas instruções para operações mais básicas)
- Uma ênfase na otimização do *pipeline de instruções*.  
(A arquitetura RISC oferece maior facilidade na implementação do *pipeline*)

# Fundamentos RISC

A implementação de *pipeline de instruções* em computadores melhora substancialmente o desempenho na execução de programas. Para isso, várias propriedades aplicadas ao conjunto de instruções implementado no processador permitem que o *pipeline* alcance, ainda mais, bons resultados. Em especial, os fundamentos de conjuntos RISC têm uma relação estreita com as técnicas de *pipeline* estudadas neste material didático.

Conceitos RISC importantes para a implementação de *pipeline*:

- Instruções de tamanho fixo (em *bits*):
  - ▶ Facilita o alinhamento de memória;
  - ▶ Facilita a busca de instruções pela CPU.
- Poucos modos de endereçamento:
  - ▶ Facilita a decodificação da instrução;
  - ▶ Facilita a busca de operandos;
- Definição de instruções simples:
  - ▶ Ciclos de clock por instrução equilibrados;
  - ▶ Facilidade na temporização do *pipeline*

# Fundamentos RISC

A implementação de *pipeline de instruções* em computadores melhora substancialmente o desempenho na execução de programas. Para isso, várias propriedades aplicadas ao conjunto de instruções implementado no processador permitem que o *pipeline* alcance, ainda mais, bons resultados. Em especial, os fundamentos de conjuntos RISC têm uma relação estreita com as técnicas de *pipeline* estudadas neste material didático.

Conceitos RISC importantes para a implementação de *pipeline*:

- Instruções de tamanho fixo (em *bits*):
  - ▶ Facilita o alinhamento de memória;
  - ▶ Facilita a busca de instruções pela CPU.
- Poucos modos de endereçamento:
  - ▶ Facilita a decodificação da instrução;
  - ▶ Facilita a busca de operandos;
- Definição de instruções simples:
  - ▶ Ciclos de clock por instrução equilibrados;
  - ▶ Facilidade na temporização do *pipeline*

# Fundamentos RISC

A implementação de *pipeline de instruções* em computadores melhora substancialmente o desempenho na execução de programas. Para isso, várias propriedades aplicadas ao conjunto de instruções implementado no processador permitem que o *pipeline* alcance, ainda mais, bons resultados. Em especial, os fundamentos de conjuntos RISC têm uma relação estreita com as técnicas de *pipeline* estudadas neste material didático.

Conceitos RISC importantes para a implementação de *pipeline*:

- Instruções de tamanho fixo (em *bits*):
  - ▶ Facilita o alinhamento de memória;
  - ▶ Facilita a busca de instruções pela CPU.
- Poucos modos de endereçamento:
  - ▶ Facilita a decodificação da instrução;
  - ▶ Facilita a busca de operandos;
- Definição de instruções simples:
  - ▶ Ciclos de clock por instrução equilibrados;
  - ▶ Facilidade na temporização do *pipeline*

# Fundamentos RISC

A implementação de *pipeline de instruções* em computadores melhora substancialmente o desempenho na execução de programas. Para isso, várias propriedades aplicadas ao conjunto de instruções implementado no processador permitem que o *pipeline* alcance, ainda mais, bons resultados. Em especial, os fundamentos de conjuntos RISC têm uma relação estreita com as técnicas de *pipeline* estudadas neste material didático.

Conceitos RISC importantes para a implementação de *pipeline*:

- Instruções de tamanho fixo (em *bits*):
  - ▶ Facilita o alinhamento de memória;
  - ▶ Facilita a busca de instruções pela CPU.
- Poucos modos de endereçamento:
  - ▶ Facilita a decodificação da instrução;
  - ▶ Facilita a busca de operandos;
- Definição de instruções simples:
  - ▶ Ciclos de clock por instrução equilibrados;
  - ▶ Facilidade na temporização do *pipeline*

# Fundamentos RISC

A implementação de *pipeline de instruções* em computadores melhora substancialmente o desempenho na execução de programas. Para isso, várias propriedades aplicadas ao conjunto de instruções implementado no processador permitem que o *pipeline* alcance, ainda mais, bons resultados. Em especial, os fundamentos de conjuntos RISC têm uma relação estreita com as técnicas de *pipeline* estudadas neste material didático.

Conceitos RISC importantes para a implementação de *pipeline*:

- Instruções de tamanho fixo (em *bits*):
  - ▶ Facilita o alinhamento de memória;
  - ▶ Facilita a busca de instruções pela CPU.
- Poucos modos de endereçamento:
  - ▶ Facilita a decodificação da instrução;
  - ▶ Facilita a busca de operandos;
- Definição de instruções simples:
  - ▶ Ciclos de clock por instrução equilibrados;
  - ▶ Facilidade na temporização do *pipeline*

# Fundamentos RISC

A implementação de *pipeline de instruções* em computadores melhora substancialmente o desempenho na execução de programas. Para isso, várias propriedades aplicadas ao conjunto de instruções implementado no processador permitem que o *pipeline* alcance, ainda mais, bons resultados. Em especial, os fundamentos de conjuntos RISC têm uma relação estreita com as técnicas de *pipeline* estudadas neste material didático.

Conceitos RISC importantes para a implementação de *pipeline*:

- Instruções de tamanho fixo (em *bits*):
  - ▶ Facilita o alinhamento de memória;
  - ▶ Facilita a busca de instruções pela CPU.
- Poucos modos de endereçamento:
  - ▶ Facilita a decodificação da instrução;
  - ▶ Facilita a busca de operandos;
- Definição de instruções simples:
  - ▶ Ciclos de clock por instrução equilibrados;
  - ▶ Facilidade na temporização do *pipeline*

# Fundamentos RISC

A implementação de *pipeline de instruções* em computadores melhora substancialmente o desempenho na execução de programas. Para isso, várias propriedades aplicadas ao conjunto de instruções implementado no processador permitem que o *pipeline* alcance, ainda mais, bons resultados. Em especial, os fundamentos de conjuntos RISC têm uma relação estreita com as técnicas de *pipeline* estudadas neste material didático.

Conceitos RISC importantes para a implementação de *pipeline*:

- Instruções de tamanho fixo (em *bits*):
  - ▶ Facilita o alinhamento de memória;
  - ▶ Facilita a busca de instruções pela CPU.
- Poucos modos de endereçamento:
  - ▶ Facilita a decodificação da instrução;
  - ▶ Facilita a busca de operandos;
- Definição de instruções simples:
  - ▶ Ciclos de clock por instrução equilibrados;
  - ▶ Facilidade na temporização do *pipeline*

# Fundamentos RISC

A implementação de *pipeline de instruções* em computadores melhora substancialmente o desempenho na execução de programas. Para isso, várias propriedades aplicadas ao conjunto de instruções implementado no processador permitem que o *pipeline* alcance, ainda mais, bons resultados. Em especial, os fundamentos de conjuntos RISC têm uma relação estreita com as técnicas de *pipeline* estudadas neste material didático.

Conceitos RISC importantes para a implementação de *pipeline*:

- Instruções de tamanho fixo (em *bits*):
  - ▶ Facilita o alinhamento de memória;
  - ▶ Facilita a busca de instruções pela CPU.
- Poucos modos de endereçamento:
  - ▶ Facilita a decodificação da instrução;
  - ▶ Facilita a busca de operandos;
- Definição de instruções simples:
  - ▶ Ciclos de clock por instrução equilibrados;
  - ▶ Facilidade na temporização do *pipeline*

# Fundamentos RISC

A implementação de *pipeline de instruções* em computadores melhora substancialmente o desempenho na execução de programas. Para isso, várias propriedades aplicadas ao conjunto de instruções implementado no processador permitem que o *pipeline* alcance, ainda mais, bons resultados. Em especial, os fundamentos de conjuntos RISC têm uma relação estreita com as técnicas de *pipeline* estudadas neste material didático.

Conceitos RISC importantes para a implementação de *pipeline*:

- Instruções de tamanho fixo (em *bits*):
  - ▶ Facilita o alinhamento de memória;
  - ▶ Facilita a busca de instruções pela CPU.
- Poucos modos de endereçamento:
  - ▶ Facilita a decodificação da instrução;
  - ▶ Facilita a busca de operandos;
- Definição de instruções simples:
  - ▶ Ciclos de clock por instrução equilibrados;
  - ▶ Facilidade na temporização do *pipeline*

# Fundamentos RISC

## Exemplos de arquiteturas RISC e não-RISC

**Tabela 13.1** Características de alguns processadores CISC, RISC e superescalares

	Computadores com conjuntos de instruções complexos (CISC)			Computadores com conjuntos de instruções reduzidos (RISC)		Superescalares		
Característica	IBM 370/168	VAX 11/780	Intel 80486	SPARC	MIPS R4000	Power PC	Ultra SPARC	MIPS R10000
Ano de desenvolvimento	1973	1978	1989	1987	1991	1993	1996	1996
Número de instruções	208	303	235	69	94	225		
Tamanho de instrução em bytes	2–6	2–57	1–11	4	4	4	4	4
Modos de endereçamento	4	22	11	1	1	2	1	1
Número de registradores de uso geral	16	16	8	40–520	32	32	40–520	32
Tamanho da memória de controle (Kb)	420	480	246	—	—	—	—	—
Tamanho da cache (KB)	64	64	8	32	128	16–32	32	64

# Pipeline de Instruções

Apresentamos um ciclo de instruções básico da Arquitetura MIPS, baseada na arquitetura RISC (Reg/Reg). (diferente do apresentado na Aula-12!)

## Ciclo de instrução do MIPS:

- ① Busca de instrução (envolve o cálculo de endereço da instrução);
- ② Decodificação de instrução (interpretação dos *bits* relativos ao *opcode* e busca os operandos);
- ③ Execução da operação definida pelo *opcode* da instrução;
- ④ Acesso para escrita e leitura na memória (Carga e Armazenamento);
- ⑤ Gravação de resultados das operações.

Se implementarmos este ciclo de instrução em forma de *pipeline* baseado na Arquitetura de Von Newmann, haverá uma concorrência pelo acesso à área de armazenamento em: (1), (2), (4) e (5)



# Pipeline de Instruções

Apresentamos um ciclo de instruções básico da Arquitetura MIPS, baseada na arquitetura RISC (Reg/Reg). (diferente do apresentado na Aula-12!)

## Ciclo de instrução do MIPS:

- ① Busca de instrução (envolve o cálculo de endereço da instrução);
- ② Decodificação de instrução (interpretação dos *bits* relativos ao *opcode* e busca os operandos);
- ③ Execução da operação definida pelo *opcode* da instrução;
- ④ Acesso para escrita e leitura na memória (Carga e Armazenamento);
- ⑤ Gravação de resultados das operações.

Se implementarmos este ciclo de instrução em forma de *pipeline* baseado na Arquitetura de Von Newmann, haverá uma concorrência pelo acesso à área de armazenamento em: (1), (2), (4) e (5)



# Pipeline de Instruções

Apresentamos um ciclo de instruções básico da Arquitetura MIPS, baseada na arquitetura RISC (Reg/Reg). (diferente do apresentado na Aula-12!)

## Ciclo de instrução do MIPS:

- ① **Busca de instrução** (envolve o cálculo de endereço da instrução);
- ② **Decodificação de instrução** (interpretação dos *bits* relativos ao *opcode* e busca os operandos);
- ③ Execução da operação definida pelo *opcode* da instrução;
- ④ **Acesso para escrita e leitura na memória** (Carga e Armazenamento);
- ⑤ **Gravação de resultados das operações.**

Se implementarmos este ciclo de instrução em forma de *pipeline* baseado na Arquitetura de Von Newmann, haverá uma concorrência pelo acesso à área de armazenamento em: (1), (2), (4) e (5)



# Pipeline de Instruções

Como a Arquitetura MIPS é (Reg/Reg), refinamos as concorrências nas áreas de armazenamento:

## Ciclo de instrução do MIPS:

- ① Busca de instrução na **memória** (envolve o cálculo de endereço da instrução);
- ② Decodificação de instrução (interpretação dos *bits* relativos ao *opcode* e busca os operandos no **banco de registradores**);
- ③ Execução da operação definida pelo *opcode* da instrução;
- ④ Acesso para escrita e leitura na **memória** (Carga e Armazenamento);
- ⑤ Gravação de resultados das operações no **banco de registradores**.

Agora temos conflitos isolados entre os grupos [(1) e (4)] e [(2) e (5)]

# Pipeline de Instruções

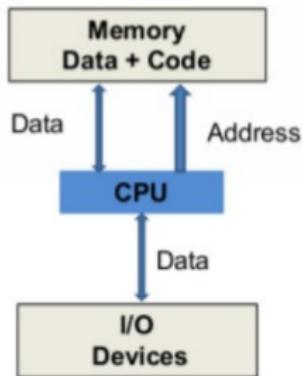
Como a Arquitetura MIPS é (Reg/Reg), refinamos as concorrências nas áreas de armazenamento:

## Ciclo de instrução do MIPS:

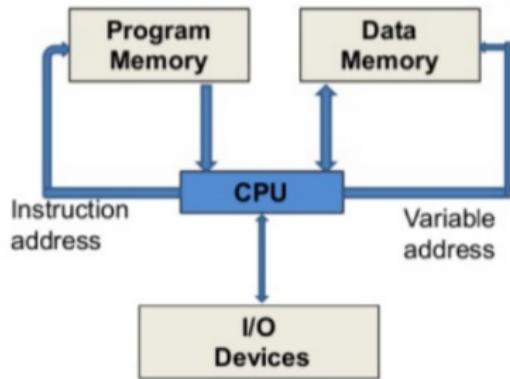
- ① Busca de instrução na **memória** (envolve o cálculo de endereço da instrução);
- ② Decodificação de instrução (interpretação dos *bits* relativos ao *opcode* e busca os operandos no **banco de registradores**);
- ③ Execução da operação definida pelo *opcode* da instrução;
- ④ Acesso para escrita e leitura na **memória** (Carga e Armazenamento);
- ⑤ Gravação de resultados das operações no **banco de registradores**.

Agora temos conflitos isolados entre os grupos [(1) e (4)] e [(2) e (5)]

# Arquitetura: Von Neumann & Arquitetura Harvard



Von Neumann Machine



Harvard Machine

# Arquitetura: Von Newmann & Arquitetura Harvard

A alteração entre as arquiteturas traz duas consequências imediatas:

- A arquitetura de Harvard permite acesso simultâneo à memória de dados e à memória de instruções;
  - ▶ Essa propriedade é importante para o *pipeline*, que sobrepõe *ciclos de instrução* em etapas distintas do ciclo.
- Os compiladores devem compilar códigos para alocação em uma única memória ou em duas memórias distintas. Esse problema pode ser solucionado alternativamente pelo sistema operacional.
  - ▶ Compiladores e sistemas operacionais de servidores e computadores pessoais geram e carregam código em memória principal única;
  - ▶ Usar a arquitetura de Von Newmann permite melhor aproveitamento da memória, sendo que tanto o código quanto os dados podem dividir o uso da memória. No Caso da arquitetura de Harvard, a alocação de dados não pode aproveitar áreas da memória de instrução disponíveis.

# Arquitetura: Von Newmann & Arquitetura Harvard

A alteração entre as arquiteturas traz duas consequências imediatas:

- A arquitetura de Harvard permite acesso simultâneo à memória de dados e à memória de instruções;
  - ▶ Essa propriedade é importante para o *pipeline*, que sobrepõe *ciclos de instrução* em etapas distintas do ciclo.
- Os compiladores devem compilar códigos para alocação em uma única memória ou em duas memórias distintas. Esse problema pode ser solucionado alternativamente pelo sistema operacional.
  - ▶ Compiladores e sistemas operacionais de servidores e computadores pessoais geram e carregam código em memória principal única;
  - ▶ Usar a arquitetura de Von Newmann permite melhor aproveitamento da memória, sendo que tanto o código quanto os dados podem dividir o uso da memória. No Caso da arquitetura de Harvard, a alocação de dados não pode aproveitar áreas da memória de instrução disponíveis.

# Arquitetura: Von Newmann & Arquitetura Harvard

A alteração entre as arquiteturas traz duas consequências imediatas:

- A arquitetura de Harvard permite acesso simultâneo à memória de dados e à memória de instruções;
  - ▶ Essa propriedade é importante para o *pipeline*, que sobrepõe *ciclos de instrução* em etapas distintas do ciclo.
- Os compiladores devem compilar códigos para alocação em uma única memória ou em duas memórias distintas. Esse problema pode ser solucionado alternativamente pelo sistema operacional.
  - ▶ Compiladores e sistemas operacionais de servidores e computadores pessoais geram e carregam código em memória principal única;
  - ▶ Usar a arquitetura de Von Newmann permite melhor aproveitamento da memória, sendo que tanto o código quanto os dados podem dividir o uso da memória. No Caso da arquitetura de Harvard, a alocação de dados não pode aproveitar áreas da memória de instrução disponíveis.

# Arquitetura: Von Newmann & Arquitetura Harvard

A alteração entre as arquiteturas traz duas consequências imediatas:

- A arquitetura de Harvard permite acesso simultâneo à memória de dados e à memória de instruções;
  - ▶ Essa propriedade é importante para o *pipeline*, que sobrepõe *ciclos de instrução* em etapas distintas do ciclo.
- Os compiladores devem compilar códigos para alocação em uma única memória ou em duas memórias distintas. Esse problema pode ser solucionado alternativamente pelo sistema operacional.
  - ▶ Compiladores e sistemas operacionais de servidores e computadores pessoais geram e carregam código em memoria principal única;
  - ▶ Usar a arquitetura de Von Newmann permite melhor aproveitamento da memória, sendo que tanto o código quanto os dados podem dividir o uso da memória. No Caso da arquitetura de Harvard, a alocação de dados não pode aproveitar áreas da memória de instrução disponíveis.

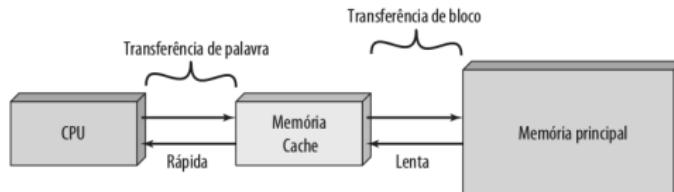
# Arquitetura: Von Newmann & Arquitetura Harvard

A alteração entre as arquiteturas traz duas consequências imediatas:

- A arquitetura de Harvard permite acesso simultâneo à memória de dados e à memória de instruções;
  - ▶ Essa propriedade é importante para o *pipeline*, que sobrepõe *ciclos de instrução* em etapas distintas do ciclo.
- Os compiladores devem compilar códigos para alocação em uma única memória ou em duas memórias distintas. Esse problema pode ser solucionado alternativamente pelo sistema operacional.
  - ▶ Compiladores e sistemas operacionais de servidores e computadores pessoais geram e carregam código em memória principal única;
  - ▶ Usar a arquitetura de Von Newmann permite melhor aproveitamento da memória, sendo que tanto o código quanto os dados podem dividir o uso da memória. No Caso da arquitetura de Harvard, a alocação de dados não pode aproveitar áreas da memória de instrução disponíveis.

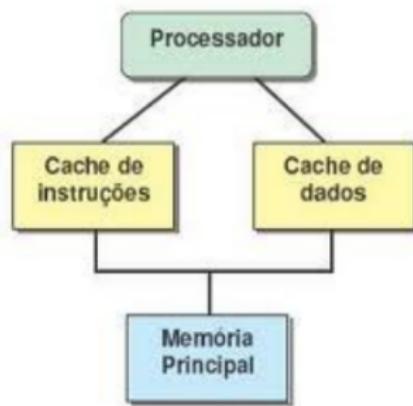
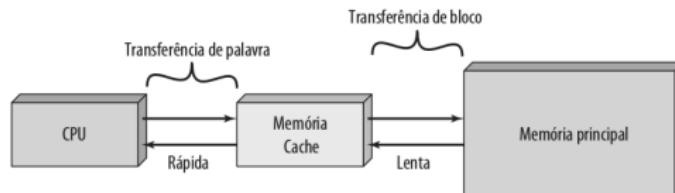
# Arquitetura: Von Newmann & Arquitetura Harvard

Computadores pessoais e servidores possuem *pipeline* e arquitetura baseada em programa armazenado de Von Newmann. Isso é eficiente?



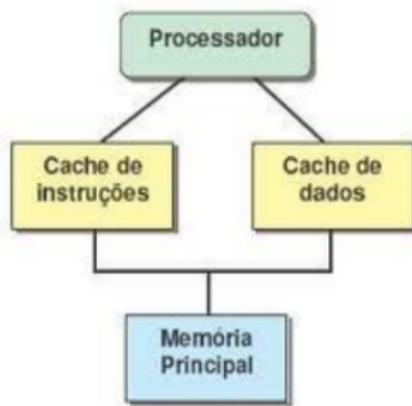
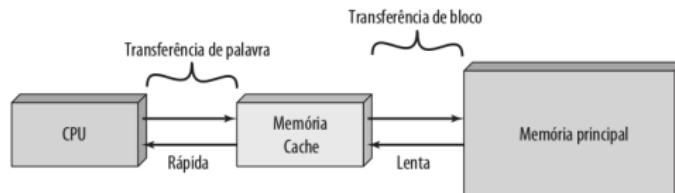
# Arquitetura: Von Newmann & Arquitetura Harvard

Computadores pessoais e servidores possuem *pipeline* e arquitetura baseada em programa armazenado de Von Newmann. Isso é eficiente?



# Arquitetura: Von Newmann & Arquitetura Harvard

Computadores pessoais e servidores possuem *pipeline* e arquitetura baseada em programa armazenado de Von Newmann. Isso é eficiente?



Solução **híbrida** (Von Newmann & Harvard).

- Sem alteração nos compiladores ou S.O.
- A gerenciadora de memória mapeia as caches.
- Memória principal assume atribuições além do auxílio na execução de programas apenas.
- CPU acessa dados e instruções simultaneamente nas caches.

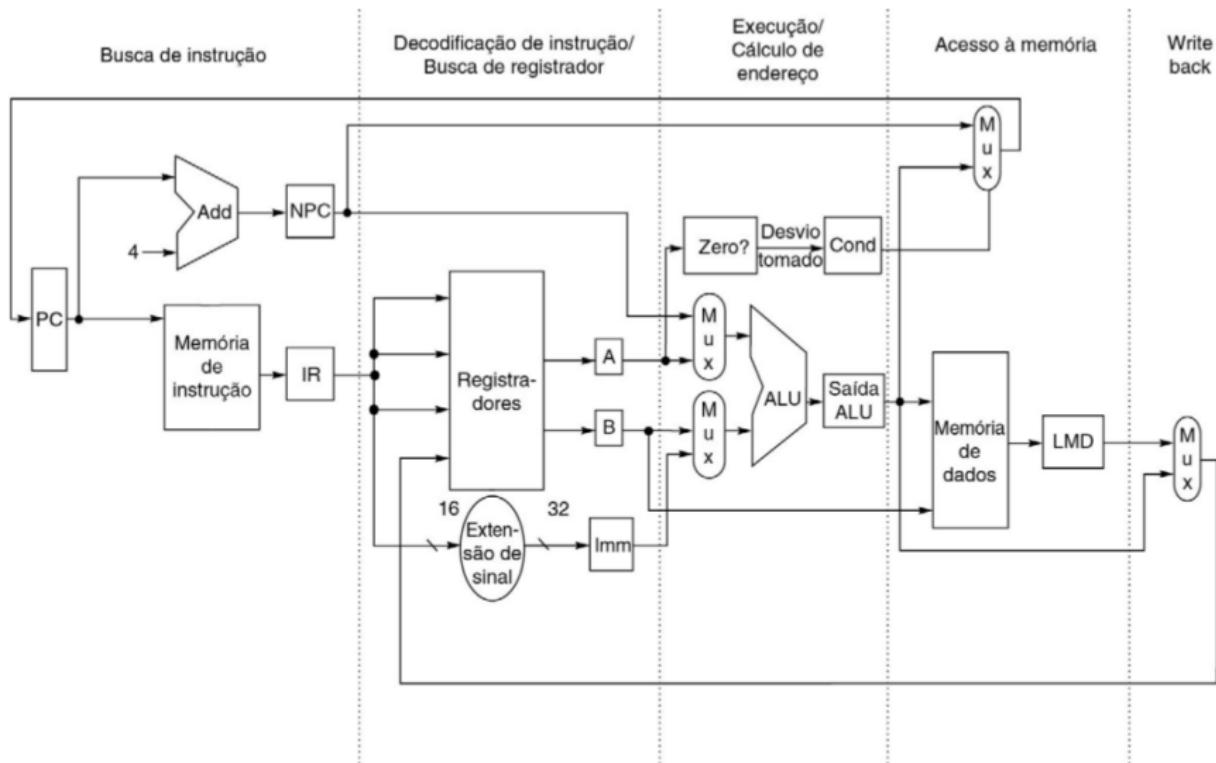
# Uma Implementação Simples de RISC

Definimos cinco estágios básico para nossa implementação. Isso não quer dizer que os *pipelines* comerciais são implementados apenas com estes estágios. Contudo, esta é uma boa referência para compreensão sobre *pipelines*.

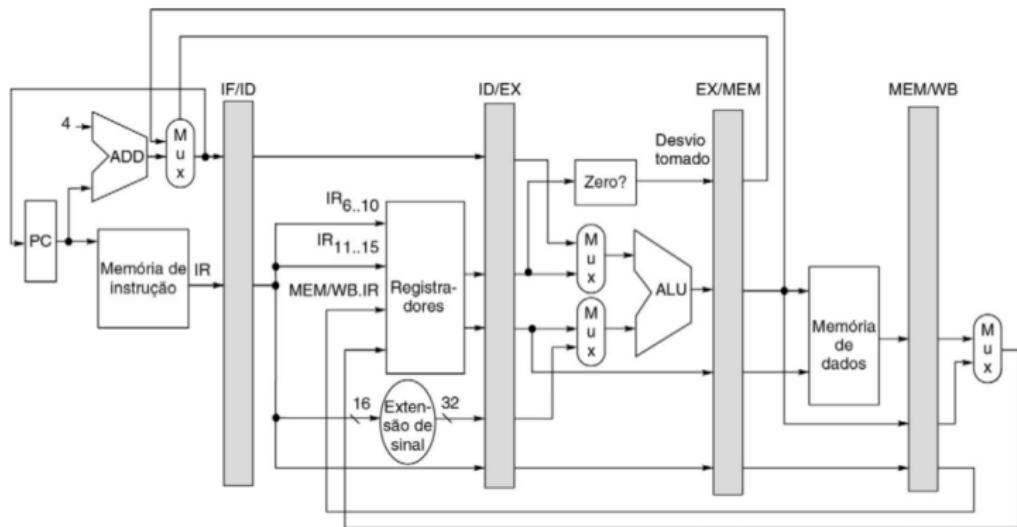
## Implementação baseada em ISA-RISC e *Load/Stor*

- ① Ciclo de busca de instrução [Instruction fetch (IF)];
- ② Ciclo de decodificação de instrução [Instruction decode (ID)];
- ③ Ciclo de execução de instrução [Instruction execution (EX)];
- ④ Acesso à memória [Memory access (MEM)];
- ⑤ Escrita de resultado [Write-back cycle (WB)];

# Uma Implementação Simples de RISC



# Um Pipeline MIPS



**FIGURA C.22** O datapath é implementado com pipeline pela inclusão de um conjunto de registradores, um entre cada par de estágios de pipe.

Os registradores servem para conduzir valores e informações de controle de um estágio para o próximo. Também podemos pensar no PC como um registrador de pipeline, que fica antes do estágio IF do pipeline, levando a um registrador de pipeline para cada estágio de pipe. Lembre-se de que o PC é um registrador acionado pela borda, escrito ao final do ciclo de clock; daf não existir condição de race (corrida) na escrita do PC. O multiplexador de seleção para o PC foi movido de modo que o PC seja escrito exatamente em um estágio (IF). Se não o movessemos, haveria um conflito quando ocorresse um desvio, pois duas instruções tentariam escrever valores diferentes no PC. A maior parte das vias de dados flui da esquerda para a direita, que é de um ponto anterior no tempo para outro posterior. As vias fluindo da direita para a esquerda (que transportam a informação de write-back do registrador e a informação do PC em um desvio) introduzem complicações ao nosso pipeline.

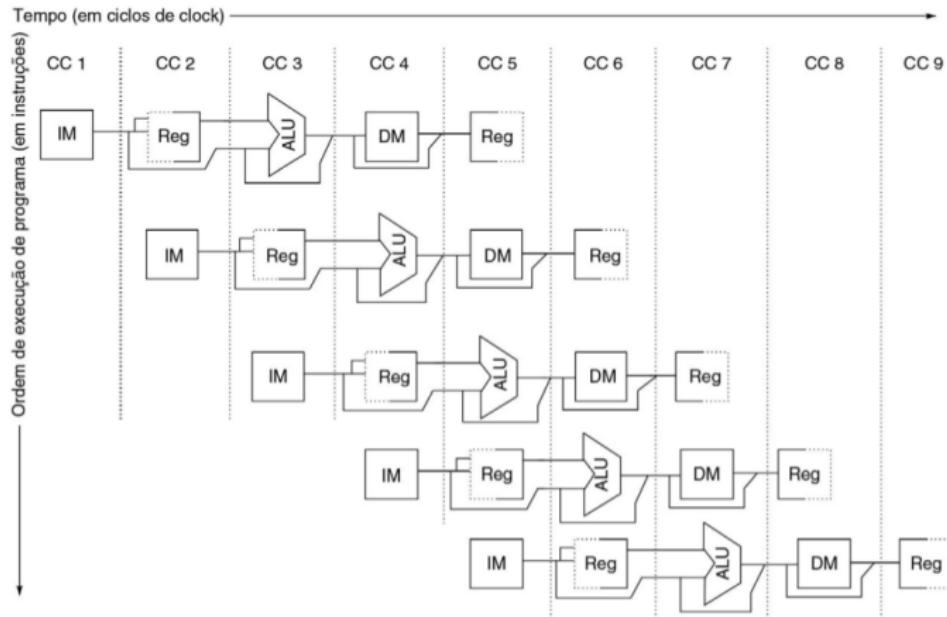
# Pipeline: Paralelismo no uso dos estágios

Número da instrução	Número do clock								
	1	2	3	4	5	6	7	8	9
Instrução $i$	IF	ID	EX	MEM	WB				
Instrução $i + 1$		IF	ID	EX	MEM	WB			
Instrução $i + 2$			IF	ID	EX	MEM	WB		
Instrução $i + 3$				IF	ID	EX	MEM	WB	
Instrução $i + 4$					IF	ID	EX	MEM	WB

**FIGURA C.1** Pipeline RISC simples.

A cada ciclo de clock, outra instrução é lida e inicia sua execução em cinco ciclos. Se uma instrução for iniciada a cada ciclo de clock, o desempenho será até cinco vezes o de um processador sem pipeline. Os nomes para os estágios no pipeline são iguais aos usados para os ciclos na implementação sem pipeline: IF = instruction fetch (busca de instrução), ID = instruction decode (decodificação de instrução), EX = execução, MEM = acesso à memória e WB = write-back.

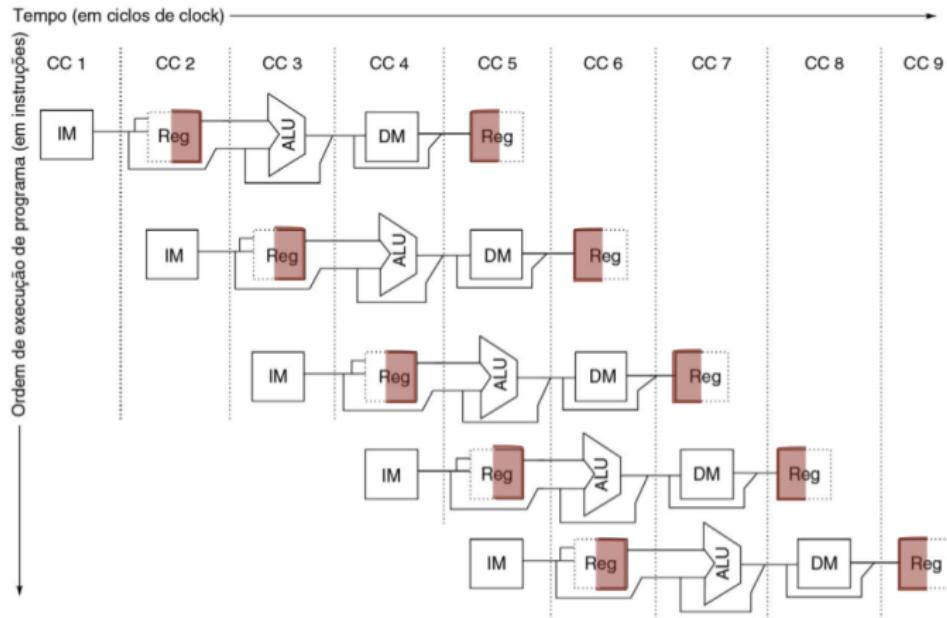
# Pipeline: Acesso ao banco de registradores



**FIGURA C.2** O pipeline pode ser imaginado como uma série de datapaths deslocados no tempo.

Isso mostra a sobreposição entre as partes do datapath, com o ciclo de clock 5 (CC 5) mostrando a situação de estado fixo. Como o banco de registradores é usado como uma fonte no estágio ID e como um destino no estágio WB, ele aparece duas vezes. Mostramos que ele é lido em uma parte do estágio e escrito em outra usando uma linha sólida, à direita ou à esquerda, respectivamente, e uma linha tracejada no outro lado. A abreviação IM é usada para Instruction Memory (memória de instrução), DM para Data Memory (memória de dados) e CC para ciclo de clock.

# Pipeline: Acesso ao banco de registradores



**FIGURA C.2** O pipeline pode ser imaginado como uma série de datapaths deslocados no tempo.

Isso mostra a sobreposição entre as partes do datapath, com o ciclo de clock 5 (CC 5) mostrando a situação de estado fixo. Como o banco de registradores é usado como uma fonte no estágio ID e como um destino no estágio WB, ele aparece duas vezes. Mostramos que ele é lido em uma parte do estágio e escrito em outra usando uma linha sólida, à direita ou à esquerda, respectivamente, e uma linha tracejada no outro lado. A abreviação IM é usada para Instruction Memory (memória de instrução), DM para Data Memory (memória de dados) e CC para ciclo de clock.

# Obstáculos do *Pipeline* (Hazards)

Algumas situações de perigo (*hazard*) podem impedir que a próxima instrução seja executada sem que ocorra uma parada no *pipeline*. Os perigos causam aumento no *CPI* do *pipeline*. Basicamente, destacamos três situações de destas:

- Perigos estruturais, que surgem de conflitos de recursos quando o hardware não pode aceitar todas as combinações possíveis de instruções simultaneamente em execução sobreposta;
- Perigos de dados (*RAW*), que surgem quando uma instrução dependente dos resultados de uma instrução anterior de maneira que é exposta pela sobreposição de instruções no pipeline;
- Perigos de controle, que surgem no *pipeline de desvios* e outras instruções que mudam o *PC*.

No capítulo 3 do livro texto, estudaremos outros perigos expostos pelos códigos de programas. Contudo, estes outros perigos não afetam diretamente o hardware, mas principalmente o compilador.



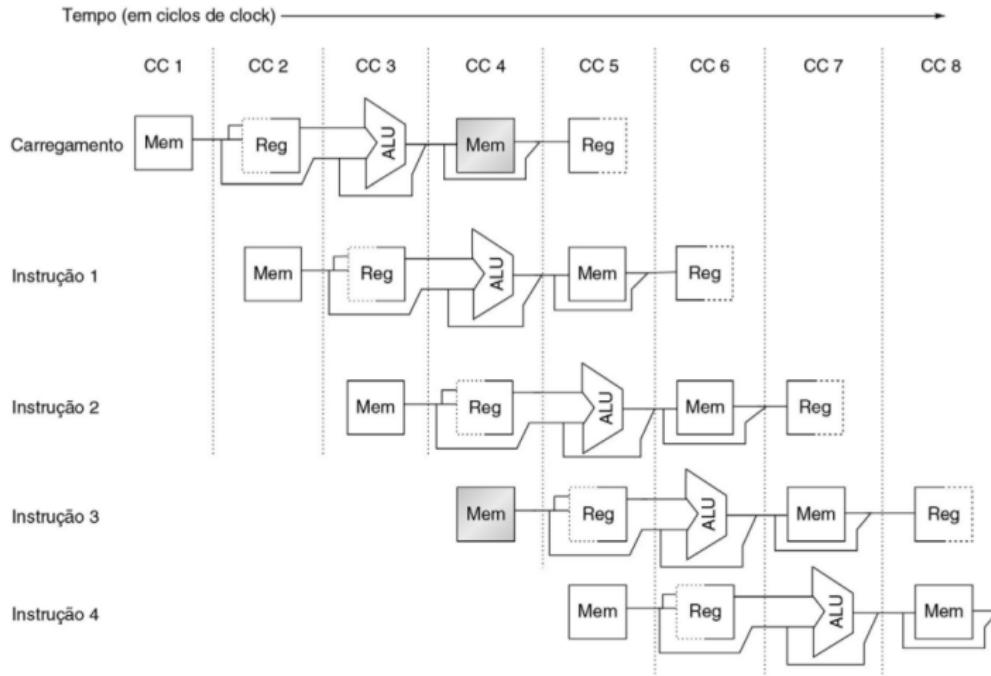
## Obstáculos do *Pipeline*: Perigos estruturais

Os perigos estruturas ocorrem quando instruções diferentes disputam os mesmos recursos em um determinado momento. Esses perigos podem acontecer, como exemplo:

- Quando o banco de registradores não permite leitura e escrita no mesmo ciclo de *clock* (como visto);
- Quando as unidades funcionais não são em forma de *pipeline* (como o caso da divisão).
- Quando há necessidade de mais de um acesso à memória no mesmo ciclo de *clock* (a seguir);

Quando ocorre um perigo estrutural, o *pipeline* se desdobra até que a unidade de hardware esteja disponível. Esses desdobramentos são com a inserção de *bolhas* ou *stall* nos estágios do *pipeline*.

# Perigo Estrutural: Conflito de acesso à memória



**FIGURA C.4** Um processador com apenas uma porta de memória gerará conflito sempre que houver uma referência de memória.

Neste exemplo, a instrução load utiliza a memória para um acesso aos dados ao mesmo tempo em que a instrução 3 deseja buscar uma instrução da memória.

# Uma Extensão para Operações Multiciclos

Instruções de ponto flutuante exigem maior *latência* para a execução. Pode-se usar mais lógica ou ciclos de *clock* mais latentes. Ao contrário, pode-se implementar as unidades funcionais também em usando os conceitos de *pipeline*

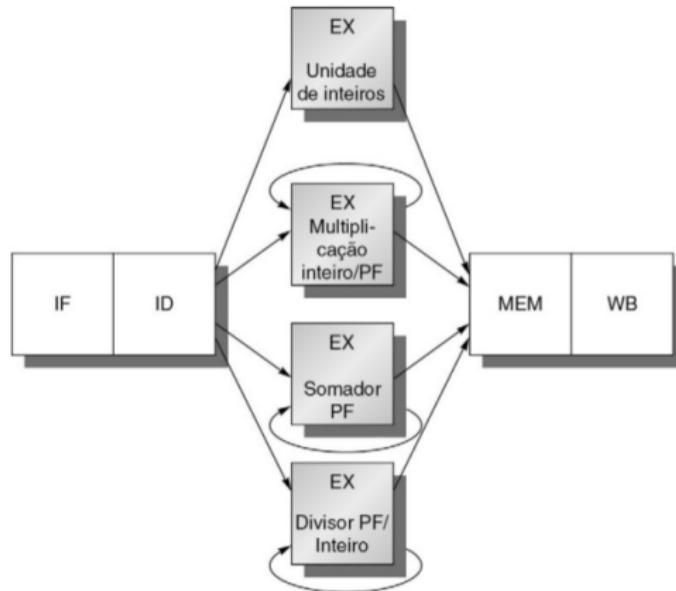
Vamos considerar quatro unidades funcionais separadas para nossa implementação MIPS:

- Unidade de inteiros: carga e armazenamento; operações de ALU com inteiros e desvios
- Multiplicador: para ponto flutuante e inteiros
- Somador de ponto flutuante: soma, subtração e conversão de ponto flutuante
- Divisor: de ponto flutuante e inteiro

# Operações Multiciclos sem *Pipeline* nas Unid. Funcionais

Unidades funcionais do MIPS sem implementação em *pipeline*

Instruções não podem ser despachadas antes do término de qualquer instrução em uma unidade funcional



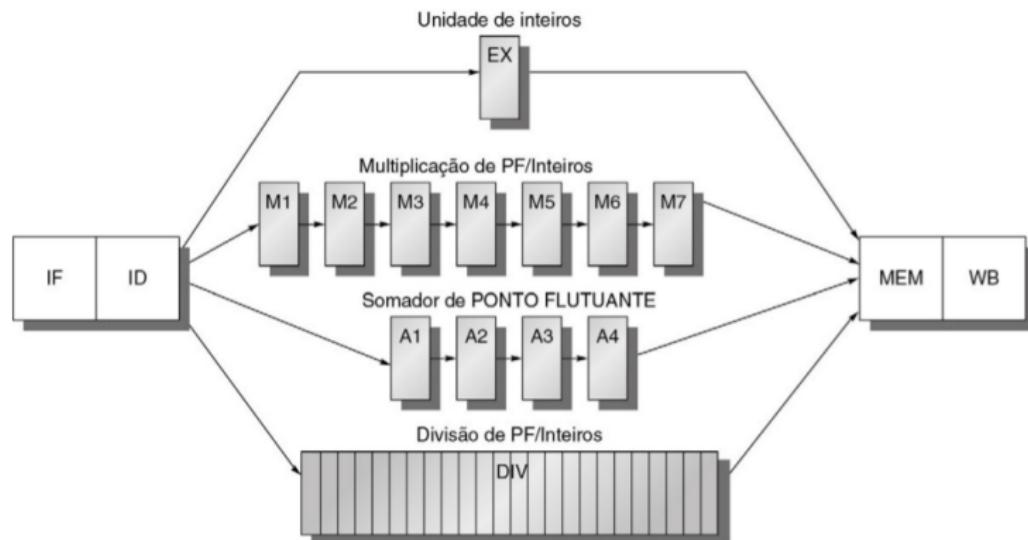
**FIGURA C.33** O pipeline MIPS com três unidades funcionais de ponto flutuante adicionais, em não pipeline.

Como somente uma instrução é despachada em cada ciclo de clock, todas as instruções passam pelo pipeline-padrão para as operações com inteiros. As operações de ponto flutuante simplesmente fazem o loop quando atingem o estágio EX. Depois de terminarem o estágio EX, elas prosseguem para MEM e WB, para concluir a execução.

# Operações Multiciclos com *Pipeline* nas Unid. Funcionais

## Unidades funcionais do MIPS sem implementação em *pipeline*

Instruções podem ser despachadas antes do término de várias instruções em unidades funcionais



**FIGURA C.35** Um pipeline que admite múltiplas operações de PF pendentes.

O multiplicador e o somador de PF estão totalmente em pipeline e possuem uma profundidade de sete e quatro estágios, respectivamente. O divisor de PF não está em pipeline, mas exige 24 ciclos de clock para concluir. A latência nas instruções entre o despacho de uma operação de PF e o uso do resultado dessa operação sem incorrer em um stall RAW é determinada pelo número de ciclos gastos nos estágios de execução. Por exemplo, a quarta instrução após uma adição de PF pode usar o resultado da adição de PF. Para operações da ALU com inteiros, a profundidade do pipeline de execução é sempre um e a próxima instrução pode usar os resultados.

# Considerações RISC & CISC

As arquiteturas CISC dominavam o cenário, mas os investimentos no RISC mudaram a realidade

Por um lado (CISC), temos grande variedade de modos de endereçamento, vasta gama de instruções de diferentes formatos e grande complexidade na implementação do hardware. Contudo, isso tudo leva à grande oferta de recursos ao desenvolvedor de linguagens e compiladores, além de gerar programas menores, economizando a memória.

Já do outro lado (RISC), a pequena variedade de modos de endereçamento e a simplicidade nas instruções de tamanho fixo facilitam a implementação do *pipeline* de instruções, que promete aumento de desempenho ao *hardware*, além da facilidade em implementá-lo. Contudo, os programas compilados tornam-se mais longos e a implementação de compiladores torna-se mais dispendiosa, necessitando mais combinações de instruções para realizar tarefas oferecidas por instruções CISC.

Com o radicalismo na definição formal de RISC e a distância destas definições aos conceitos CISC, uma variedade de arquiteturas são citadas como *híbridas*, tendo mais características CISC ou RISC. Veja, por exemplo, que para uma arquitetura ser considerada totalmente RISC, ela deve ter apenas um modo de endereçamento, o que é muito ineficiente.

Certamente, os avanços RISC trouxeram muitos benefícios ao tirar parte da atenção ao CiSC. Hoje, por exemplo, a Intel implemente um processador misto com parte CISC e outra RISC para executar um conjunto de instruções CISC, o x86. Isso traz benefícios aos processadores desta empresa.

As discussões sobre CISC & RISC são longas e, muitas vaidades e afinidades determinam a posição de pesquisadores. Para completar essa discussão, leia a seção 13.8 *Controvérsia de RISC versus CISC* no livro texto.

# Agradecimentos

## Agradecimentos Especiais:

Agradeço a toda a comunidade L<sup>A</sup>T<sub>E</sub>X.  
Em especial a *Till Tantau* pelo *Beamer*.

<https://www.tcs.uni-luebeck.de/mitarbeiter/tantau/>

Desta forma, tornou-se possível a escrita deste material didático.

## Exercícios:

Lista de exercícios divulgada no Moodle

