

Arquitetura e Organização de Computadores

Aula-12: Estrutura e Função do Processador

Eliseu César Miguel

Livro: Arquitetura e Organização de Computadores
William Stallings 8^a Edição
Universidade Federal de Alfenas

February 25, 2021



Organização da Aula

- 1 Introdução
- 2 Organização do Processador
- 3 Organização dos Registradores
- 4 Pipeline de Instruções (Parte 1)

Organização da Aula

- 1 Introdução
- 2 Organização do Processador
- 3 Organização dos Registradores
- 4 Pipeline de Instruções (Parte 1)

Organização da Aula

- 1 Introdução
- 2 Organização do Processador
- 3 Organização dos Registradores
- 4 Pipeline de Instruções (Parte 1)

Organização da Aula

- 1 Introdução
- 2 Organização do Processador
- 3 Organização dos Registradores
- 4 Pipeline de Instruções (Parte 1)

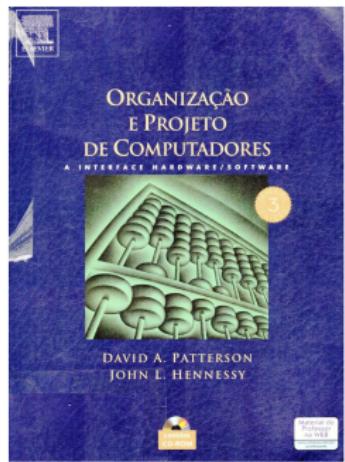
Bibliografia básica



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

Estrutura e função do processador

Os principais pontos em relação à estrutura e função do processador são:

- Organização do processador (no que envolve a execução de instruções)
- Organização dos registradores
- Ciclo da instrução
- *Pipeline de Instruções*

Muito do que veremos já foi tratado em nosso curso. O que repetiremos contribuirá para organizarmos o conhecimento!

Organização do Processador

Para entender a organização do processador, vamos considerar os requisitos que lhe são exigidos:

- **Buscar instrução:** o processador lê uma instrução da memória (registrador, cache, memória principal)
- **Interpretar a instrução:** a instrução é decodificada para determinar qual ação é requerida
- **obter os dados:** a execução de uma instrução pode requerer leitura de dados da memória ou um módulo de E/S
- **processar os dados:** a execução de uma instrução pode requerer efetuar alguma operação aritmética ou lógica com os dados
- **gravar os dados:** os resultados de uma execução podem requerer gravar dados para memória ou um módulo E/S (ou registradores)

Esses tópicos foram vistos quando estudamos *ciclo de instrução*. Aproveitamos para lembrar que há várias classes de instruções, o que gera refinamentos na lista anterior!

(Chamaremos informalmente estes requisitos por *etapas do ciclo de instrução...*)

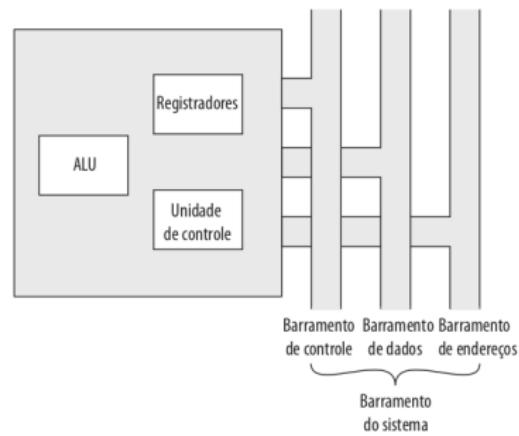
Organização do Processador

A execução das etapas citadas pode exigir que dados sejam armazenados temporariamente

- Qual é a próxima instrução?
- Onde gravar o resultado? (Lembrem-se das arquiteturas Reg/Reg)
- Onde devemos guardar a instrução buscada?
- Como armazenar temporariamente as informações geradas por cada uma das etapas do ciclo de instrução?

É necessária uma memória interna!

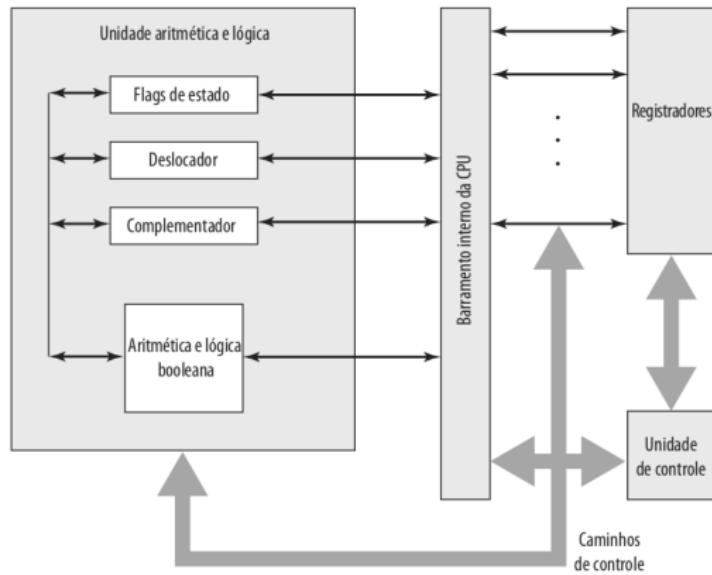
Figura 12.1 CPU com barramento de sistema



Organização do Processador

Estrutura mais detalhada da CPU

Figura 12.2 Estrutura interna da CPU



Organização dos Registradores

Em geral, na hierarquia mais alta de memória, temos:

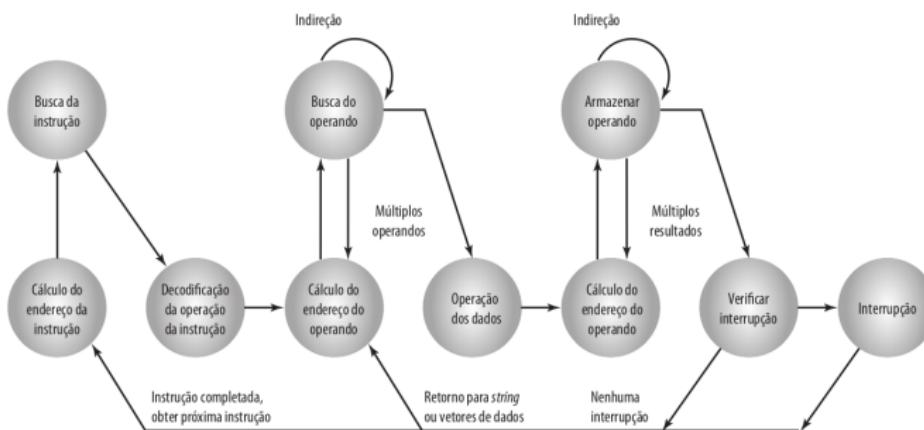
- **Registradores visíveis ao usuário:** possibilitam que o programador de linguagem de máquina ou *assembly* minimize as referências à memória, pela otimização do uso de registradores.
 - ▶ Uso geral (um banco para uso geral do programador, ou...)
 - ▶ Dados (banco especializado)
 - ▶ Endereços (banco especializado)
 - ▶ Códigos condicionais (ou *flags*)
- **Registradores de controle e estado:** usados pela unidade de controle para controlar a operação do processador e por programas privilegiados do Sistema Operacional para controlar a execução de programas:
 - ▶ Contador de programas (PC): contém o endereço de uma instrução a ser lida
 - ▶ Registrador da instrução (IR): contém a instrução lida mais recentemente
 - ▶ Registrador de endereço de memória (MAR): contém o endereço de uma posição de memória
 - ▶ Registrador de *buffer* de memória (MBR): contém uma palavra de dados para ser escrita na memória ou a palavra lida mais recentemente

Pipeline de Instruções (Parte 1)

Como os conceitos de conjunto de instruções RISC e CISC são importantes para a compreensão de *pipelines*, faremos aqui uma abordagem geral e, em seguida, estudaremos com mais detalhes na Aula-13

Considere o ciclo de instruções:

Figura 12.5 Diagrama de estado do ciclo da instrução



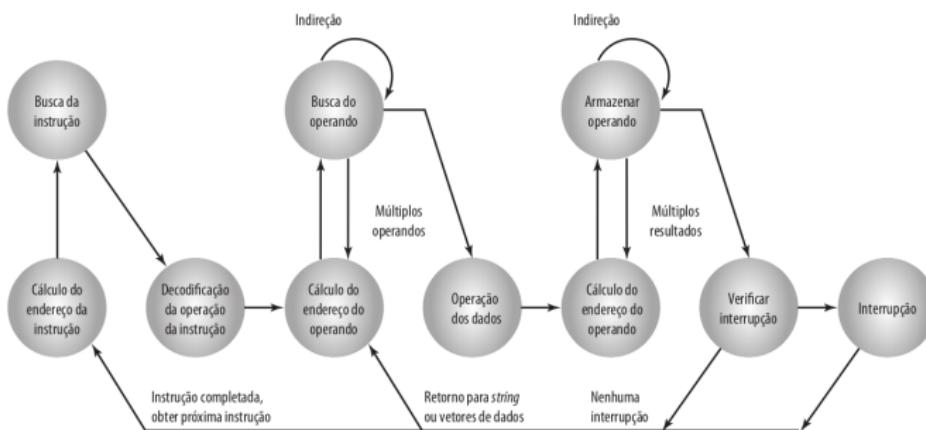
É possível buscar uma instrução i_c ao mesmo tempo em que a instrução i_b é decodificada e a instrução i_a está acessando seus operandos?

Pipeline de Instruções (Parte 1)

Como os conceitos de conjunto de instruções RISC e CISC são importantes para a compreensão de *pipelines*, faremos aqui uma abordagem geral e, em seguida, estudaremos com mais detalhes na Aula-13

Considere o ciclo de instruções:

Figura 12.5 Diagrama de estado do ciclo da instrução



É possível buscar uma instrução i_c ao mesmo tempo em que a instrução i_b é decodificada e a instrução i_a está acessando seus operandos?

Pipeline de Instruções

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.

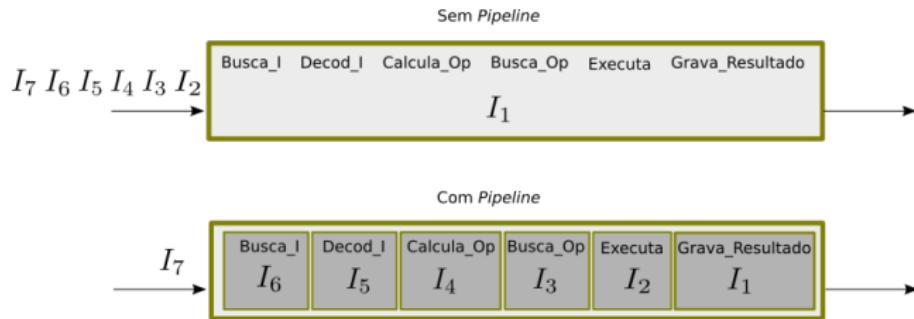
Conceitos necessários para o estudo de *pipeline*

- Fundamentos RISC
Apresentados na Aula-13
- Arquitetura: Von Newmann & Arquitetura Harvard.
Apresentado na disciplina Interface hardware/software

Pipeline de Instruções (processador escalar)

Processador sem pipeline: vários ciclos de *clock* são dedicados a uma instrução I_i até que todas as etapas do ciclo de instrução sejam realizadas. A instrução subsequente I_j só inicia sua execução ao fim do ciclo de I_i .

Processador com pipeline: as etapas são implementadas como *componentes* independentes. Quando a instrução I_i é transferida, por exemplo, da *busca* para a *decodificação*, a unidade de busca está livre para buscar a instrução I_j . Com isso, várias instruções são executadas em sobreposição temporal.



Pipeline de instruções: Características

Os *pipelines* de instrução podem apresentar diferentes quantidades e ações em suas camadas (camadas, etapas e canais são sinônimos neste contexto)

Em nosso exemplo de *pipeline*, as etapas são: (veremos outro exemplo na Aula-13)

- **Buscar instrução (FI, do inglês Fetch Instruction):** ler a próxima instrução esperada em um *buffer*.
- **Decodificar instrução (DI):** determinar o *opcode* e os especificadores dos operandos.
- **Calcular operandos (CO):** calcular o endereço efetivo de cada operando de origem. Isto pode envolver endereçamento por deslocamento, registrador indireto, indireto ou outras formas de cálculo de endereço.
- **Obter operandos (FO, do inglês Fetch Operands):** obter cada operando da memória. Operandos que estão nos registradores não precisam ser lidos da memória.
- **Executar instrução (EI):** efetuar a operação indicada e armazenar o resultado, se houver, na posição do operando de destino especificado.
- **Escrever operando (WO, do inglês Write Operands):** armazenar o resultado na memória.

Há *pipelines* com mais de 30 camadas. O que se espera é ter camadas com equilíbrio entre si de acordo com os ciclos de *clock* que cada uma necessita para executar sua tarefa.

Pipeline de instruções: Exemplo

Considere a execução de 9 instruções sem qualquer parada no *pipeline*:

Figura 12.10 Diagrama de tempo para operação do pipeline da instrução

Tempo →

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Instrução 1	FI	DI	CO	FO	EI	WO								
Instrução 2		FI	DI	CO	FO	EI	WO							
Instrução 3			FI	DI	CO	FO	EI	WO						
Instrução 4				FI	DI	CO	FO	EI	WO					
Instrução 5					FI	DI	CO	FO	EI	WO				
Instrução 6						FI	DI	CO	FO	EI	WO			
Instrução 7							FI	DI	CO	FO	EI	WO		
Instrução 8								FI	DI	CO	FO	EI	WO	
Instrução 9									FI	DI	CO	FO	EI	WO

Neste caso, consideramos a execução em desempenho teórico do *pipeline*.

Pipeline de instruções: Desempenho

Vários fatores afetam o desempenho do *pipeline*:

- O equilíbrio em ciclos de *clock* entre as camadas
pode definir a quantidade de camadas implementadas
- A dependência entre as instruções
sugere escalonamento estático ou dinâmico de instruções
- As restrições funcionais
podem influenciar na quantidade de unidades funcionais na organização do computador
- As instruções de controle
incentivou o desenvolvimento de técnicas de predição de desvios
- O conjunto de instruções
incentivou o desenvolvimento da arquitetura RISC Reg/Reg

Basicamente, a disciplina *Interface hardware/software* aprofunda estes fatores e apresenta técnicas para se implementar o *pipeline*

Pipeline de instruções: Desempenho

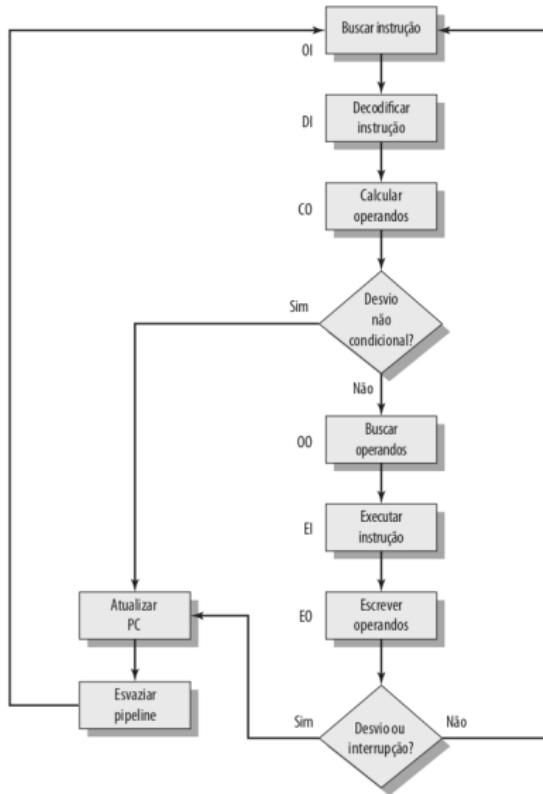
Ocorrência de instrução de desvio (instrução 3)

Figura 12.11 O efeito de um desvio condicional na operação do pipeline da instrução

	Tempo							Penalidade por desvio						
	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Instrução 1	FI	DI	CO	FO	EI	WO								
Instrução 2		FI	DI	CO	FO	EI	WO							
Instrução 3			FI	DI	CO	FO	EI	WO						
Instrução 4				FI	DI	CO	FO							
Instrução 5					FI	DI	CO							
Instrução 6						FI	DI							
Instrução 7							FI							
Instrução 15								FI	DI	CO	FO	EI	WO	
Instrução 16								FI	DI	CO	FO	EI	WO	

Pipeline de instruções: Exemplo de 6 camadas

Figura 12.12 Pipeline de instrução de uma CPU de seis estágios



Assuntos não abordados nesta apresentação

O Capítulo 13 do livro apresenta vários outros tópicos, (listados abaixo). Estes sobrepõem-se com o conteúdo da disciplina *Interface hardware/software*. Assim, para quem não pretende se matricular na disciplina que cobre esse conteúdo com mais profundidade, sugere-se a leitura do Capítulo 13 do livro texto em sua totalidade

- Desempenho do *pipeline*
- *Hazards* do *pipeline*
- Lidando com desvios
- *Pipeline* de Intel 80486

Na Aula-13 estudaremos os conceitos RISC. Assim, aproveitaremos para voltar ao estudo de *pipeline* em um exemplo baseado na arquitetura RISC do MIPS.

Agradecimentos

Agradecimentos Especiais:

Agradeço a toda a comunidade L^AT_EX.
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

