

Arquitetura e Organização de Computadores

Aula-11: Conjunto de Instruções: Modos e Formatos de Endereços

Eliseu César Miguel

Livro: Arquitetura e Organização de Computadores

William Stallings 8^a Edição

Universidade Federal de Alfenas

February 1, 2023



Organização da Aula

- 1 Introdução
- 2 Modos de endereçamento
- 3 Formatos de instruções
- 4 Linguagem de montagem

Organização da Aula

- 1 Introdução
- 2 Modos de endereçamento
- 3 Formatos de instruções
- 4 Linguagem de montagem

Organização da Aula

- 1 Introdução
- 2 Modos de endereçamento
- 3 Formatos de instruções
- 4 Linguagem de montagem

Organização da Aula

- 1 Introdução
- 2 Modos de endereçamento
- 3 Formatos de instruções
- 4 Linguagem de montagem

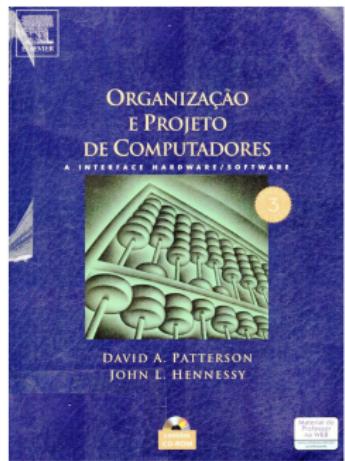
Bibliografia básica



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

Os modos de endereçamento definem como os dados podem ser acessados a partir de uma instrução

Com o aumento no tamanho das memórias e as necessidades do *software* vários modos foram propostos.

Com isso, ao decodificar uma instrução, é necessário saber qual é o modo de endereçamento de cada campo de endereço.

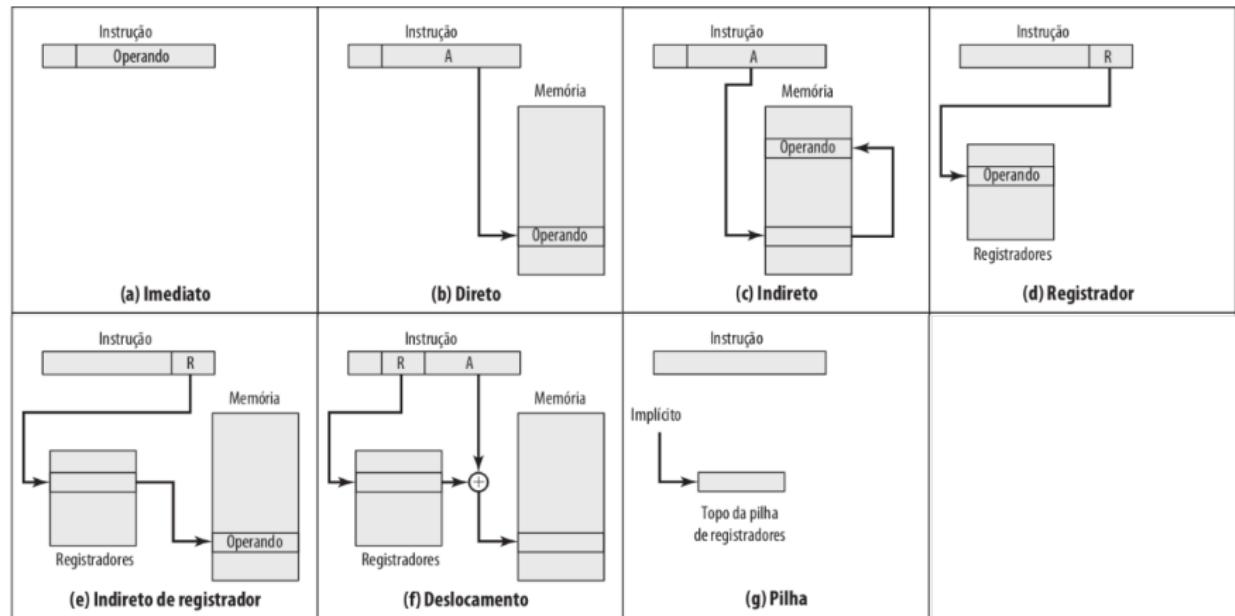
Os modos podem ser definidos no *opcod* ou em *bits* especiais.

Quanto mais modos de endereçamento a arquitetura oferece:

- mais poder ela dá ao programador (compilador)
- mais complexa torna-se a busca de operandos
- mais ciclos de instrução podem ser necessários para a execução da instrução
- RISC (poucos modos de endereçamento) maior facilidade em implementar o *hardware*
- CISC (muitos modos) maior facilidade de implementar o *software*

Modos de endereçamento

Para se acessar um operando, a CPU precisa identificar como este operando foi endereçado. Ao fim, algum cálculo matemático pode ser necessário.



Modos de endereçamento

A tabela mostra como interpretar os endereços.

Tabela 11.1 Modos básicos de endereçamento

Modo	Algoritmo	Principal vantagem	Principal desvantagem
Imediato	Operando = A	Nenhuma referência de memória	Magnitude de operando limitada
Direto	EA = A	Simples	Espaço de endereçamento limitado
Indireto	EA = (A)	Espaço de endereçamento grande	Múltiplas referências de memória
Registrador	EA = R	Nenhuma referência de memória	Espaço de endereçamento limitado
Indireto de registrador	EA = (R)	Espaço de endereçamento grande	Referência extra de memória
Deslocamento	EA = A + (R)	Flexibilidade	Complexidade
Pilha	EA = topo da pilha	Nenhuma referência de memória	Aplicabilidade limitada

- A = conteúdos de um campo de endereço dentro da instrução
- R = conteúdos de um campo de endereço na instrução que se refere a um registrador
- EA = endereço real (efetivo) do local que contém o operando referenciado
- (X) = conteúdos do local de memória X ou do registrador X

Em uma instrução, operandos diferentes podem ser endereçados por modos diferentes.

O cálculo de endereço pode aumentar a complexidade e o envolvimento de unidades funcionais específicas.



Modos de endereçamento

Modo de endereçamento	Instrução de exemplo	Significado	Quando é usado
Registrador	Add R4, R3	$\text{Regs[R4]} \leftarrow \text{Regs[R4]} + \text{Regs[R3]}$	Quando um valor está em um registrador.
Imediato	Add R4, #3	$\text{Regs[R4]} \leftarrow \text{Regs[R4]} + 3$	Para constantes.
Deslocamento	Add R4, 100(R1)	$\text{Regs[R4]} \leftarrow \text{Regs[R4]} + \text{Mem[100+Regs[R1]]}$	Ao acessar variáveis locais (+ simula modos de endereçamento indireto de registrador e direto).
Indireto de registrador	Add R4, (R1)	$\text{Regs[R4]} \leftarrow \text{Regs[R4]} + \text{Mem[Regs[R1]]}$	Ao acessar usando um ponteiro ou um endereço calculado.
Indexado	Add R3, (R1+R2)	$\text{Regs[R3]} \leftarrow \text{Regs[R3]} + \text{Mem[Regs[R1]+Regs[R2]]}$	Algumas vezes útil no endereçamento de array: R1 = base do array; R2 = valor de índice.
Direto ou absoluto	Add R1, (1001)	$\text{Regs[R1]} \leftarrow \text{Regs[R1]} + \text{Mem[1001]}$	Algumas vezes útil para acessar dados estáticos; a constante do endereço pode precisar ser grande.
Indireto de memória	Add R1, @(R3)	$\text{Regs[R1]} \leftarrow \text{Regs[R1]} + \text{Mem[Mem[Regs[R3]]]}$	Se R3 é o endereço de um ponteiro p , então o modo produz " p ".
Autoincremento	Add R1, (R2)+	$\begin{aligned} &+ \text{Mem[Regs[R2]]} \\ &\text{Regs[R2]} \leftarrow \text{Regs[R2]} + d \end{aligned}$	Útil para percorrer os arrays passo a passo dentro de um loop. R2 aponta para o início do array; cada referência incrementa R2 pelo tamanho de um elemento, d .
Autodecremento	Add R1, -(R2)	$\begin{aligned} &\text{Regs[R2]} \leftarrow \text{Regs[R2]} - d \\ &\text{Regs[R1]} \leftarrow \text{Regs[R1]} + \text{Mem[Regs[R2]]} \end{aligned}$	Mesmo uso do autoincremento. Autodecremento/autoincremento também podem agir como push/pop para implementar uma pilha.
Escalonado	Add R1, 100(R2)[R3]	$\begin{aligned} &\text{Regs[R1]} \leftarrow \text{Regs[R1]} + \text{Mem[100+Regs[R2]]} \\ &+ \text{Regs[R3]} * d \end{aligned}$	Usado para indexar arrays. Pode ser aplicado para qualquer modo de endereçamento indexado em alguns computadores.



Modos de endereçamento

O modo de endereçamento por deslocamento é muito poderoso. Permite endereçar memórias com grande número de endereços.

Neste modo de endereçamento, pelo menos dois campos são necessários, podendo um ser implícito Três exemplos de uso do modo de endereçamento por deslocamento:

- **Relativo ao PC:** Usado para deslocamentos durante a execução de um programa. Pode ser usado, por exemplo para fazer as iterações de um laço de repetição
- **Registrador base:** Combina um valor de um endereço dentro de uma página de memória, definida pelo registrador base
- **Indexação:** Usado para deslocar em acessos a partir de uma posição definida, como em um vetor

Exemplo de endereçamento:

for (int <i>i</i> = 100; <i>i</i> > 0; <i>i</i> --){	572	Loop	L.D	F0,	0(R1)	
A[<i>i</i>]=A[<i>i</i>]+k	573		ADD.D	F4,	F0,	F2
}	574		S.D	F4,	0(R1)	
	575		DADDU.I	R1,	R1,	#-8
	576		BNE	R1,	R2,	Loop

Formatos de instruções

O formato da instrução define o *layout* de bits de uma instrução, no que diz respeito aos campos que a constituem.

Principais pontos de projeto:

- **Tamanho** decisão relacionada ao tamanho da memória, organização da memória, estrutura do barramento, complexidade e velocidade do processador. Esta decisão determina a riqueza e a flexibilidade da máquina do ponto de vista do programador da linguagem de montagem.
- **Alocação de bits**: importante decisão que pode ser afetada pela quantidade de instruções e tamanho fixo ou variável

do ponto de vista de *bits* para endereçamento, temos que considerar:

- ▶ número de modos de endereçamento: pode exigir ou mais *bits*
- ▶ número de operandos: pode exigir ou mais *bits*
- ▶ Registrador versus memória: pode dificultar ter instruções de tamanho fixo
- ▶ número de conjuntos de registradores: pode diminuir os *bits* de endereços de registradores, usando conjuntos especializados de forma implícita
- ▶ Intervalo de endereços: uso de endereçamento por deslocamento para diminuir os *bits* na instrução
- ▶ granularidade do endereço: relaciona à unidade endereçável. Endereçar a memória por byte aumenta a granularidade



Formatos de instruções

O formato da instrução define o *layout* de bits de uma instrução, no que diz respeito aos campos que a constituem.

Principais pontos de projeto:

- **Tamanho** decisão relacionada ao tamanho da memória, organização da memória, estrutura do barramento, complexidade e velocidade do processador. Esta decisão determina a riqueza e a flexibilidade da máquina do ponto de vista do programador da linguagem de montagem.
- **Alocação de bits**: importante decisão que pode ser afetada pela quantidade de instruções e tamanho fixo ou variável
do ponto de vista de *bits* para endereçamento, temos que considerar:
 - ▶ **número de modos de endereçamento**: pode exigir ou mais *bits*
 - ▶ **número de operandos**: pode exigir ou mais *bits*
 - ▶ **Registrador versus memória**: pode dificultar ter instruções de tamanho fixo
 - ▶ **número de conjuntos de registradores**: pode diminuir os *bits* de endereços de registradores, usando conjuntos especializados de forma implícita
 - ▶ **Intervalo de endereços**: uso de endereçamento por deslocamento para diminuir os *bits* na instrução
 - ▶ **granularidade do endereço**: relaciona à unidade endereçável. Endereçar a memória por byte aumenta a granularidade

Linguagem de montagem

Um processador pode entender e executar instruções de máquina. Essas instruções são simples números binários armazenados no computador. Se o programador quisesse programar diretamente na linguagem de máquina

Figura 11.13 Computação na fórmula $N = I + J + K$

Endereço	Conteúdo			
101	0010	0010	101	2201
102	0001	0010	102	1202
103	0001	0010	103	1203
104	0011	0010	104	3204
201	0000	0000	201	0002
202	0000	0000	202	0003
203	0000	0000	203	0004
204	0000	0000	204	0000

(a) Programa binário

Endereço	Conteúdo
101	2201
102	1202
103	1203
104	3204
201	0002
202	0003
203	0004
204	0000

(b) Programa hexadecimal

Endereço	Instrução	
101	LDA	201
102	ADD	202
103	ADD	203
104	STA	204
201	DAT	2
202	DAT	3
203	DAT	4
204	DAT	0

(c) Programa simbólico

Rótulo	Operação	Operando
FORMUL	LDA	I
	ADD	J
	ADD	K
	STA	N
I	DATA	2
J	DATA	3
K	DATA	4
N	DATA	0

(d) Programa assembly

há erros no exemplo? Este é um desafio!

Linguagem de montagem

Um processador pode entender e executar instruções de máquina. Essas instruções são simples números binários armazenados no computador. Se o programador quisesse programar diretamente na linguagem de máquina

Figura 11.13 Computação na fórmula $N = I + J + K$

Endereço	Conteúdo			
101	0010	0010	101	2201
102	0001	0010	102	1202
103	0001	0010	103	1203
104	0011	0010	104	3204
201	0000	0000	201	0002
202	0000	0000	202	0003
203	0000	0000	203	0004
204	0000	0000	204	0000

(a) Programa binário

Endereço	Conteúdo
101	2201
102	1202
103	1203
104	3204
201	0002
202	0003
203	0004
204	0000

(b) Programa hexadecimal

Endereço	Instrução	
101	LDA	201
102	ADD	202
103	ADD	203
104	STA	204
201	DAT	2
202	DAT	3
203	DAT	4
204	DAT	0

(c) Programa simbólico

Rótulo	Operação	Operando
FORMUL	LDA	I
	ADD	J
	ADD	K
	STA	N
I	DATA	2
J	DATA	3
K	DATA	4
N	DATA	0

(d) Programa assembly

há erros no exemplo? Este é um desafio!

Linguagem de montagem

Um processador pode entender e executar instruções de máquina. Essas instruções são simples números binários armazenados no computador. Se o programador quisesse programar diretamente na linguagem de máquina

Figura 11.13 Computação na fórmula $N = I + J + K$

End	Conteúdo				
101	0010	0010	0000	0001	
102	0001	0010	0000	0010	
103	0001	0010	0000	0011	
104	0011	0010	0000	0100	
201	0000	0000	0000	0001	
202	0000	0000	0000	0010	
203	0000	0000	0000	0011	
204	0000	0000	0000	0100	

(a) Programa binário

Endereço	Conteúdo
101	2201
102	1202
103	1203
104	3204
201	0002
202	0003
203	0004
204	0000

(b) Programa hexadecimal

Endereço	Instrução	
101	LDA	201
102	ADD	202
103	ADD	203
104	STA	204
201	DAT	2
202	DAT	3
203	DAT	4
204	DAT	0

(c) Programa simbólico

Rótulo	Operação	Operando
FORMUL	LDA	I
	ADD	J
	ADD	K
	STA	N
I	DATA	2
J	DATA	3
K	DATA	4
N	DATA	0

(d) Programa assembly

há erros no exemplo? Este é um desafio!

Linguagem de montagem

Para saber mais sobre linguagens de montagens, veja o Apêndice B do livro texto

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

