

Introdução à Arquitetura e Organização de Computadores

Professor: Diego de Assis Monteiro Fernandes

Aula 6 - Instruções e Estilos de Endereçamento MIPS

Como já foi visto nas aulas anteriores, arquitetura MIPS apresenta, em suas instruções, formas diferentes de acessar dados. Essas formas são chamadas de **estilos de endereçamento**. Um dos estilos já vistos, chamado **endereçamento de registrador**, consiste em acessar o dado diretamente no registrador, por exemplo com a instrução `add $s0, $s1, $s2`. Outra forma de acessar um dado é obtê-lo da memória. Para fazer isso, o endereço de memória que contém o dado é obtido através da soma do endereço base (valor contido em um registrador) com o deslocamento (uma constante fornecida na instrução). Esse esquema é chamado **endereçamento base**.

A terceira forma de endereçamento já vista é utilizada pela instrução `j`, que desloca o fluxo de execução para um endereço especificado na constante embutida na instrução. Esse esquema é chamado **endereçamento pseudo-direto**, uma vez que o valor do registrador PC é alterado pela constante contida na instrução. Para compreender melhor esse último esquema descrito, é preciso compreender qual formato de instrução é utilizado por `j`. Esse formato, chamado tipo-J, é detalhado nesta aula.

1 Outros estilos de endereçamento MIPS

Existem mais duas formas de endereçamento na arquitetura MIPS. Estas formas são descritas a seguir, apresentando também outras instruções da arquitetura.

1.1 Constante ou Operandos Imediatos

Essa seção discute o endereçamento imediato.

1.1.1 Instruções Aritméticas Imediatas

Utilizando somente as instruções apresentadas até agora, nenhuma operação aritmética poderia ser realizada diretamente com constantes (os exemplo com `add` e `sub` utilizando constantes eram adaptados). Para realizar operações aritméticas com constantes, instruções específicas que incluem constantes foram criadas. Essas constantes são chamadas **imediatas** e o formato de tais instruções é o tipo-I (*immediate*), onde o campo que contém a constante é o de 16 *bits*. O estilo de endereçamento nesses casos é chamado **endereçamento imediato**.

Instrução de adição com operando imediato:

```
addi $sp,$sp,4
```

O campo `op` para `addi` é 8. A instrução com o código de máquina é a seguinte:

op	rs	rt	immediate
8	29	29	4

O código de máquina em binário para a instrução:

op	rs	rt	immediate
001000	11101	11101	0000 0000 0000 0100

1.1.2 Instrução de Comparação Imediata

As constantes imediatas também são úteis em comparações, pois é comum que um programa implementado em uma linguagem de alto nível, compare variáveis com constantes. Para comparar valores contido em registradores com constantes imediatas, uma versão imediata da instrução `slt` está presente na arquitetura. A instrução `slti` está mostrada a seguir.:

```
slti $t0, $s2,10 # $t0 = 1 se $s2 < 10
```

1.1.3 Vantagem do endereçamento Imediato

Endereçamento imediato ilustra o último princípio de projeto de hardware:

Princípio de projeto 4:

Tornar rápido o que é comum.

Constantes são frequentemente utilizadas como operandos. Por isso, fazer com que as constantes sejam parte das instruções, é uma forma de tornar a execução das instruções mais rápida do que se a constante fosse buscada na memória.

1.1.4 A instrução lui

Como já foi dito, as instruções que permitem o endereçamento imediato utilizam com formato o tipo-I. Esse tipo limita a representação de uma constante em 16 *bits*, que é o tamanho do campo reservado. Porém, é comum que, em programas de alto nível, constantes tenham valores maiores do que o máximo permitido em uma representação de 16 bits. Para solucionar tal limitação, o conjunto de instruções MIPS inclui a instrução **lui** (*load upper immediate*), especialmente desenvolvida para armazenar uma constante nos 16 bits mais altos de um registrador. Dessa forma, uma constante que excede a limitação pode ser decomposta em duas partes de 16 bits e, em seguida, atribuída com auxílio da instrução `lui` a um registrador.

A sintaxe dessa instrução é a seguinte:

```
lui $t0, 255
```

Esse exemplo atribui aos 16 bits de maior valor do registrador `$t0`, o valor 255.

A instrução `lui` também utiliza o formato tipo-I. Para ilustrar o preenchimento das campos para essa instrução, a versão em linguagem de máquina de `lui $t0,255` é mostrado a seguir.

op	rs	rt	immediate
001111	00000	01000	0000 0000 1111 1111

Como pode-se notar, a representação binária de 255 em 16 *bits* é 0000 0000 1111 1111. Com isso, o conteúdo do registrador `$t0` depois de executar `lui $t0,255` é mostrado a seguir.

Parte alta (16 bits)	Parte baixa (16 bits)
0000 0000 1111 1111	0000 0000 0000 0000

O exemplo a seguir mostra a decomposição de uma constante de 32 *bits* em duas porções de 16 *bits*. A constante é então armazenada no registrador `$s0` utilizando a instrução `lui`.

Exemplo:

Qual é o código de montagem MIPS para armazenar a seguinte constante de 32 bits no registrador `$s0`?

0000 0000 0011 1101 0000 1001 0000 0000

Primeiro os 16 *bits* mais altos são armazenados. O valor 0000 0000 0011 1101 em binário é correspondente a 61 em decimal.

lui \$s0, #61 decimal = 0000 0000 0011 1101 binário

O próximo passo é adicionar os 16 bits mais baixos, cujo valor decimal é 2304:

addi \$s0, \$s0, 2304

Para que o montador trate casos como o do exemplo mostrado, onde constantes maiores de 16 bits são utilizadas em operações¹, o registrador **\$at** é reservado pelo montador e utilizado como temporário.

1.2 Endereçamentos em Desvios

Existem duas formas de endereçamento para as instruções de desvios.

1.2.1 Endereçamento pseudo-direto e o Formato Tipo-J

A forma mais simples de endereçamento é encontrada na instrução **j**. Essa instrução utiliza o último formato ainda não apresentado, chamado tipo-J, que consiste de 6 *bits* para o campo de operação e os 26 *bits* restantes para o campo de endereço. Assim, a instrução

j 10000

é montada em linguagem de máquina no seguinte formato:

2	10000
6 bits (opcode)	26 bits (address)

A instrução **j** utiliza o estilo de **endereçamento pseudo-direto**, onde o operando é encontrado nos 26 *bits* da instrução. Para que o fluxo de execução seja alterado, a instrução atribui o operando ao registrador PC. O pseudo, contido no nome do endereçamento, deve-se ao fato da constante possuir somente 26 *bits* em contraste com os 32 bits do registrador.

1.2.2 Endereçamento relativo

Diferentemente da instrução **j**, que é uma instrução de desvio incondicional, as instruções de desvio condicional devem especificar dois operandos além do endereço de desvio. A instrução **bne** é uma instrução de desvio condicional e utiliza o formato tipo-I. Assim,

bne \$s0,\$s1,Exit

é montada no seguinte formato, onde 16 bits são reservados para o endereço:

5	16	17	10000
6 bits	5 bits	5 bits	16 bits (address)

Observando a instrução pode-se argumentar o seguinte: O que o campo de 16 *bits* deve armazenar? Uma primeira solução seria utilizá-lo para armazenar o endereço do desvio. Porém, isso limitaria o espaço de endereçamento do programa em 16 *bits* e, com isso, nenhum programa poderia conter mais do que 2¹⁶ instruções. Esse valor é muito pequeno se comparado ao tamanho da maioria dos programas atuais.

Como já foi observado, as instruções de desvio condicional são utilizadas para transformar estruturas de linguagens de alto nível como **if**, **for** e **while**. Tais estruturas desviam a execução para instruções que estão muito próxima a instrução executada. A partir desse comentário, uma alternativa para computar o endereço de desvio seria especificar, além da constante contida na instrução, um registrador que sempre seria adicionado ao campo. Em resumo, essa alternativa propõe que o endereço de desvio seja calculado como segue:

¹Essas constantes necessitam ser divididas e transferidas para um registrador, pois uma instrução suporta no máximo constantes de 16 bits

PC = Registrador + Endereço

Esta soma permite ao programa ser maior do que 2^{32} , mantendo o formato tipo-I para a instrução. A questão é qual registrador utilizar na adição? Novamente: tais desvios são geralmente utilizados em laços ou testes condicionais, desviando assim o fluxo para endereços próximos à instrução corrente. Por isso, o valor do registrador **PC** é adequado, pois contém o endereço da instrução adjacente à instrução em execução.

Essa forma de endereçamento é chamada de **endereçamento relativo** e é utilizada pela arquitetura MIPS. O relativo deve-se ao fato de que a constante armazenada nas instruções de desvio condicional não contém o endereço do desvio. Ao invés disso, essa constante especifica o desvio relativo ao valor já existente no registrador PC.

1.2.3 Formato da instrução jal

Já a instrução **jal**, utilizada para invocar procedimentos, pode desviar a execução para endereços distantes da instrução corrente. Nesse caso, como o único operando necessário é o endereço de desvio, a instrução **jal** utiliza o formato tipo-J.

1.2.4 Exemplo

Observe o laço *while* em linguagem de montagem mostrado a seguir:

```
Loop: add $t1,$s3,$s3 #temp reg $t1 = 2 * i
      add $t1,$t1,$t1 #temp reg $t1 = 4 * i
      add $t1,$t1,$s5 #t1 = endereço de save[i]
      lw $t0, 0($t1) #temp reg $t0 = save[i]
      bne $t0,$s5,Exit #vá para Exit se save[i] é diferente de k
      add $s3,$s3,$s4 #i = i + j
      j Loop #vá para Loop
Exit:
```

Qual é o código de máquina MIPS para o trecho mostrado, assumindo que o laço inicia na posição 8000 da memória?

8000	0	19	19	9	0	32
8004	0	9	9	9	0	32
8008	0	9	21	9	0	32
8012	35	9	8			0
8016	5	8	21			8
8020	0	19	20	19	0	32
8024	2					80000
8028						...

Observação: Uma vez que todas as instruções MIPS possuem o tamanho de 4 palavras, MIPS utiliza no endereçamento relativo o número de palavras e não o número de *bytes*. Assim o campo de endereço da instrução **bne** no exemplo anterior deveria ser 2 ao invés de 8. O mesmo acontece com o endereço de 26 *bits* para o formato tipo-J.

2 Resumo dos Modos de Endereçamento MIPS

A Tabela 1 lista as instruções MIPS já apresentadas, descrevendo seus respectivos formatos.

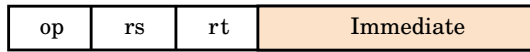
Como foi descrito, uma instrução pode acessar um operando de várias formas. Essas múltiplas formas de endereçamento são genericamente denominadas **modos de endereçamento**. Os modos de endereçamento MIPS foram descritos durante as aulas e estão resumidos nos itens a seguir e ilustrados na Figura 1.

Instrução	Nome	Formato
add	add	R
subtract	sub	R
add imediate	addi	I
load word	lw	I
store word	sw	I
load byte	lb	I
store byte	sb	I
load upper imediate	lui	I
branch if equal	beq	I
branch if not equal	bne	I
set less than	slt	R
set less than imediate	slti	I
jump	j	J
jump register	j	R
jump and link	jal	J

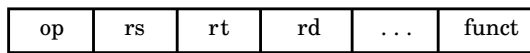
Tabela 1: Instruções MIPS apresentadas (até agora) e seus formatos.

- **Endereçamento de registrador**, onde o operando é um registrador.
- **Endereçamento base**, onde o operando está na memória, cujo endereço é a soma de um registrador com uma constante da instrução.
- **Endereçamento imediato**, onde o operando é uma constante contida na instrução.
- **Endereçamento relativo (ao PC)**, onde o endereço é a soma do registrador PC com uma constante da instrução.
- **Endereçamento pseudo-direto**, onde o endereço de 26 bits da instrução é concatenado com os bits mais altos de PC.

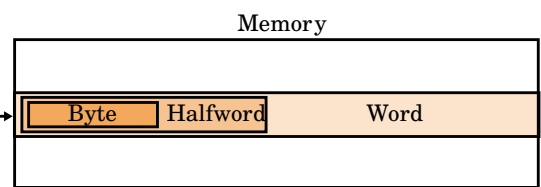
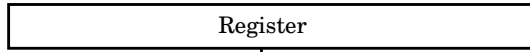
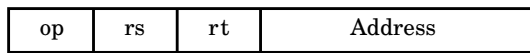
1. Immediate addressing



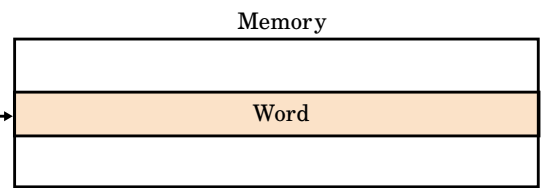
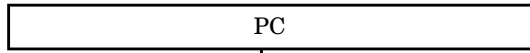
2. Register addressing



3. Base addressing



4. PC-relative addressing



5. Pseudodirect addressing

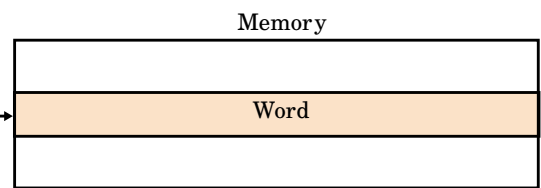
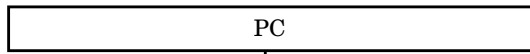
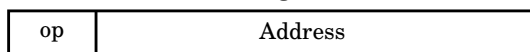


Figura 1: Ilustração dos cinco modos de endereçamento MIPS.