

Introdução à Arquitetura e Organização de Computadores

Professor: Diego de Assis Monteiro Fernandes

Aula 4 - Instruções: a linguagem da máquina

1 Introdução

As palavras compreendidas por uma máquina, ou seja, as palavras da linguagem de uma máquina são chamadas **instruções**, e seu vocabulário é chamado de **conjunto de instruções**. O objetivo dessa aula é mostrar um conjunto de instruções e verificar como ele é representado no *hardware* e a relação com as linguagens de programação de alto nível. O conjunto de instruções escolhido é da arquitetura MIPS, utilizada pela NEC, Nintendo, Silicon Graphics, Sony, e outros. É um conjunto de instruções típico projetado desde o início dos anos 80.

2 Operações do hardware do computador

Para iniciar o estudo das operações básicas de um computador é interessante escolher um tipo que seja fundamental para todas as arquiteturas. Todo computador deve ser capaz de realizar operações aritméticas, como soma, subtração, multiplicação e divisão de números. A notação em linguagem de montagem da arquitetura MIPS para uma instrução aritmética é a seguinte:

```
add a, b, c
```

Essa instrução informa ao computador para somar as variáveis b e c e armazenar o resultado da soma em a . A notação das instruções aritméticas MIPS possui esse formato que permite a realização de somente uma operação por instrução, necessitando sempre de três variáveis. O exemplo a seguir ilustra um caso simples onde algumas instruções são agrupadas para somar quatro valores.

Exemplo:

Somar b, c, d, e e armazenar a soma em a .

A sequência de código especificada realiza a soma das quatro variáveis.

```
add a, b, c
add a, a, d
add a, a, e
```

Estabelecer que todas as instruções tenham exatamente três operandos, facilita o projeto do *hardware*, deixando-o mais simples. O projeto de um *hardware* para instruções que permitem um número de operandos variável é mais complicado do que um *hardware* para um número fixo de operandos.

Princípio de projeto 1:

A simplicidade é favorecida pela regularidade.

O exemplo a seguir ilustra a utilização das duas instruções aritméticas apresentadas.

Exemplo:

Traduzir o trecho de código em C mostrado a seguir para MIPS:

```
a = b + c;
d = a - e;
```

Em MIPS:

```
add a, b, c
sub d, a, e
```

O próximo exemplo demonstra a tradução de um exemplo um pouco mais complexo de operações aritméticas em C para a linguagem de montagem. O intuito desse exemplo é verificar que em baixo nível pode-se utilizar variáveis temporárias para armazenar alguns pré-resultados.

Exemplo:

Traduzir o trecho de código um pouco mais complexo em C mostrado a seguir para MIPS:

```
f = (g + h) - (i + j);
```

O compilador deve quebrar esse trecho de código em diversas instruções de montagem, uma vez que as instruções MIPS só realizam uma única operação. Para isso serão utilizadas as variáveis temporárias $t0$ e $t1$.

```
add t0, g, h
add t1, i, j
sub f, t0, t1
```

3 Operandos do hardware de computador

Diferente dos programas escritos em linguagens de alto nível, os operandos das instruções aritméticas não podem ser quaisquer variáveis como as que foram utilizadas nos exemplos já mostrados. Os operandos devem fazer parte de um conjunto limitado de estruturas denominadas **registradores**. Os registradores são os “tijolos” da construção de um computador, pois são as primitivas usadas no projeto de *hardware* que continuam visíveis ao programador quando o computador está completo. O tamanho de um registrador na arquitetura MIPS é de *32bits*. Além disso, a arquitetura manipula seus dados como grupos de *32bits* e por isso esses agrupamentos são chamados de **palavras**(word)).

A maior diferença entre as variáveis de uma linguagem de programação e os registradores é a de que os registradores tem a sua quantidade limitada. *A arquitetura MIPS possui 32 registradores*. A razão em limitar em 32 registradores pode ser justificada no segundo princípio básico de projeto de hardware.

Princípio de projeto 2:

Quanto menor, mais rápido.

Embora seja possível escrever instruções utilizando os números dos registradores (de 0 à 31), MIPS convencionou em usar dois caracteres seguido do sinal \$ para representar um registrador. Os registradores \$s0, \$s1, ... são utilizados para corresponderem às variáveis declaradas em um programa em C e os registradores \$t0, \$t1, ... são utilizados para armazenar dados temporários (como mostrado no exemplo anterior).

Para o exemplo anterior $f = (g + h) - (i + j)$, o código MIPS seria o seguinte:

```
add $t0, $s1, $s2
add $t1, $s3, $s4
sub $s0, $t0, $t1
```

Categoria	Instrução	Exemplo	Significado	Comentário
Aritmética	adição	add a, b, c	$a = b + c$	Sempre três operandos
	subtração	sub a, b, c	$a = b - c$	Sempre três operandos

Tabela 1: Instruções aritméticas MIPS

Se o número de registradores é limitado, então como o computador consegue representar estruturas de dados mais complexas, como por exemplo vetores? A resposta para tal questão é simples: *Essas estruturas são mantidas na memória*. Entretanto, como já foi mencionado, as instruções aritméticas da arquitetura só executam suas operações manipulando valores contido em registradores. Por isso, MIPS também deve incluir instruções que sejam capazes de transferir dados da memória para registradores e vice-versa. Estas são chamadas **instruções de transferência de dados**.

Para acessar uma palavra na memória, a instrução deve fornecer um **endereço de memória**. Através desse endereço pode-se então acessar o valor desejado. O esquema de endereçamento da memória é similar ao esquema para acessar elementos de um grande vetor de dados, onde o endereço atua como o índice do vetor, iniciando de 0. A Figura 1 ilustra o que foi descrito.

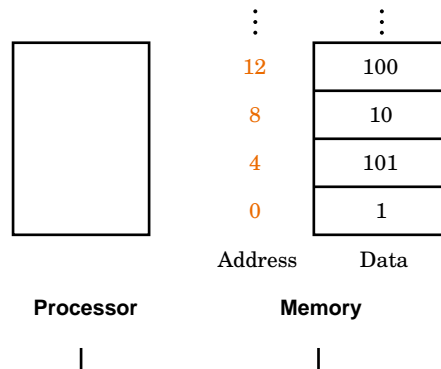


Figura 1: Endereçamento de memória em MIPS e seus elementos agrupados em palavras de 32 bits (que correspondem a 4 bytes).

A instrução de transferência de dados da memória para os registradores é tradicionalmente chamada *load*. Seu formato é o seu nome seguido pelo registrador a ser atualizado e então uma constante e um registrador que são combinados para formar o endereço de uma posição da memória. O endereço de memória é formado pela soma da parte constante da instrução e o conteúdo armazenado pelo segundo registrador. O nome dessa instrução em MIPS é **lw** (*load word*) e seu formato é o seguinte: **lw registrador1, constante(registrador2)**, onde o endereço de memória será igual a *constante* mais o conteúdo do *registrador2*. O exemplo a seguir ilustra uma utilização da instrução **lw**.

Exemplo:

Traduza o seguinte código em C para MIPS, assumindo que A é um vetor de 100 palavras e que o compilador associou as variáveis g e h com os registradores \$s1 e \$s2, respectivamente. Além disso, assumo que o endereço base do vetor é \$s3.

```
g = h + A[8];
```

O código em MIPS correspondente é o seguinte:

```
lw $t0, 8($s3)
```

Essa operação armazena o elemento A[8] do vetor em \$t0.

```
add $s1, $s2, $t0
```

Por fim a instrução **add** realiza a soma e armazena o resultado no registrador esperado. A constante utilizada na instrução de transferência de dados é chamada **offset** (deslocamento) e o registrador que possui o endereço de memória é chamado **registrador de base**, pois esse endereço é adotado como uma base para encontrar o endereço efetivo do dado a ser buscado ($endereço = base + offset$).

A instrução complementar a instrução de load é tradicionalmente denominada *store* e transfere dados de um registrador para a memória. A instrução MIPS que realiza tal operação é a **sw** (*store word*) e possui o mesmo formato da instrução **lw**. O exemplo a seguir ilustra a utilização das duas instruções de transferência de dados disponíveis na arquitetura. É importante notar que as constantes utilizadas nas instruções são referentes aos índices dos vetores. Porém, os índices aparecem no código MIPS multiplicados por 4, uma vez em que os endereços da memória são referentes a *bytes* e a arquitetura trabalha com palavras de 4 bytes (ver Figura 1).

Exemplo:

Qual o código MIPS para o seguinte código em C:

```
A[12] = h + A[8];
```

O código será o seguinte:

```
lw $t0, 32($s3)
add $t0, $s2, $t0
```

Por fim armazena o resultado:

```
sw = $t0, 48($s3)
```

Exercício: Se for necessário acessar um elemento $A[i]$ de um vetor, o valor de i deve ser multiplicado por 4 para ser utilizado no código MIPS. Suponha que i esteja no registrador **\$s0**, como multiplicá-lo por 4 utilizando instruções de soma?

4 Representando instruções no computador

Essa seção mostra a diferença entre a nossa visão de uma instrução de baixo nível e a visão de um computador. Nosso modo de raciocínio está organizado de acordo com os números da base 10, enquanto que o computador utiliza a base binária que é composta somente dos algarismos 0 e 1. A justificativa para isso é a de que os dados são representados como um conjunto de *bits* que, em baixo nível, são modelados através de sinais elétricos, baixo ou alto, vistos logicamente como 0 e 1. Instruções também são mantidas no computador como uma série de sinais elétricos e, com isso, podem ser representadas como números. Na realidade, uma instrução pode ser considerada como um agrupamento de números individuais. Nesse caso, ela é formada pela colocação destes números em sequência.

Uma vez que os registradores são utilizados em quase todas as instruções, deve haver uma convenção para mapear nomes de registradores em números. Na linguagem de montagem MIPS, os **registradores \$s0 à \$s7** são mapeados de 16 à 23 e os **registradores \$t0 à \$t7** são mapeados de 8 à 15. O exemplo a seguir converte uma instrução MIPS em linguagem de montagem para a linguagem de máquina, ou seja, mostra o seu código binário correspondente.

Exemplo:

Como é representada em conjunto de bits a instrução?

```
add $t0, $s1, $s2
```

Sua representação decimal é:

0	17	18	8	0	32
---	----	----	---	---	----

Pode-se ver então pelo exemplo que a instrução **add** é composta por um conjunto de 6 segmentos distintos. Cada um desses segmentos de uma instrução é denominado **campo**. O primeiro e último campo (0 e 32) dessa instrução combinados informam ao computador que esta instrução realiza uma adição. O segundo campo, com o valor 17, informa o número do primeiro registrador operando da soma (\$s1) e o terceiro campo, com valor 18, informa o número do segundo operando da soma (\$s2). O quarto campo, com valor 8, contém o número do registrador que recebe o resultado da soma, no caso o registrador \$t0. O quinto campo não é utilizado nessa instrução e por isso tem o seu valor em 0. A instrução também pode ser representada como campos de números binários, como mostrado a seguir.

000000	10001	10010	01000	00000	100000
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

Para distinguir a instrução em seu formato binário de seu formato em linguagem de montagem, a versão numérica das instruções é chamada de **linguagem de máquina** e uma sequência de tais instruções é chamada de **código de máquina**. Como é possível notar, através do exemplo em formato binário, a instrução MIPS mostrada possui exatamente *32 bits* (o tamanho de uma palavra na arquitetura).

4.1 Campos das instruções MIPS

Os nomes dados aos campos de uma instrução aritmética são mostrados na Figura 2.

op	rs	rt	rd	shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

Figura 2: Formato tipo-R de uma instrução MIPS.

- **op**: operação básica da instrução, tradicionalmente chamada **opcode**;
- **rs**: o primeiro registrador operando;
- **rt**: o segundo registrador operando;
- **rd**: o registrador de destino da operação;
- **shamt**: deslocamento (o termo será explicado posteriormente);
- **funct**: função, esse campo seleciona a variante específica da operação especificada no campo **op**.

Um problema ocorre quando uma instrução necessita de campos maiores do que os mostrados acima, onde o maior campo tem o tamanho de 6 bits. Por exemplo, a instrução *lw* (load word) deve especificar dois registradores e uma constante. Se a constante for armazenada em um dos campos de *5bits*, seu valor será limitado em 2^5 ou 32. Como a constante é utilizada para compor o endereço de um dado a ser selecionado na memória, o valor deve ser maior do que 32. 32 permitiria deslocar apenas 8 palavras a partir do endereço base e, dado o tamanho das memórias atuais, o campo de 5 bits é muito pequeno.

Essa questão leva a um conflito entre o desejo de manter todas as instruções com o mesmo tamanho e o desejo de manter um único formato de instrução, que constituem as duas soluções imediatas ao impasse. Tal conflito está relacionado com o terceiro princípio de projeto.

Princípio de projeto 3:

Bons projetos demandam bons compromissos.

O compromisso dos projetistas da arquitetura MIPS era o de manter todas as instruções do mesmo tamanho, ou seja, com 32 bits, requerindo assim diferentes formatos de instruções. O formato já apresentado é chamado **tipo-R** (referente a registrador). Um segundo tipo de formato de instrução é chamado **tipo-I** e é utilizado pelas instruções de transferência de dados. Seu formato é mostrado na Figura 3.

op	rs	rt	address
6 bits	5 bits	5 bits	16 bits

Figura 3: Formato tipo-I de uma instrução MIPS.

Os 16 bits são dedicados à constante que compõe o endereço. Isso permite que uma instrução de *load word* indique qualquer palavra (word) dentro de uma região de $\pm 2^{15}$ ou 32768 bytes (2^{13} ou 8192 palavras) partindo do endereço indicado no registrador de base (**rs**). É importante notar que **rt** indica o registrador de destino nesse formato de instrução.

Exemplo: Quais são os valores dos campos para a seguinte instrução:

```
lw $t0, 32($s3)
```

19 (\$s3) para o campo rs, 8 (\$t0) para o campo rt e 32 para o campo address.

Embora múltiplos formatos de instrução compliquem o hardware, é possível reduzir a complexidade utilizando a idéia aplicada em MIPS: **estabelecer formatos similares**. Por exemplo, os primeiros três campos dos formatos tipo-I e tipo-R são do mesmo tamanho e possuem o mesmo nome. Dessa forma é possível distinguir os formatos através do primeiro campo **op**. A tabela 2 mostra os números utilizados em cada campo para as instruções MIPS já vistas.

instrução	formato	op	rs	rt	rd	shamt	funct	address
add	R	0	reg	reg	reg	0	32	—
sub	R	0	reg	reg	reg	0	34	—
lw	I	35	reg	reg	—	—	—	endereço
sw	I	43	reg	reg	—	—	—	endereço

Tabela 2: Instruções MIPS e seus campos

Exemplo: O seguinte trecho de código em C:

```
A[300] = h + A[300];
```

pode ser representado pelas instruções:

```
lw $t0, 1200($t1)
add $t0, $s2, $t0
sw $t0, 1200($t1)
```

Qual é o código de máquina para essas três instruções?

op	rs	rt	address		
			rd	shamt	funct
35	9	8	1200		
0	18	8	8	0	32
43	9	8	1200		

O equivalente em binário para as instruções é o seguinte:

op	rs	rt	address		
			rd	shamt	funct
100001	01001	01000	0000 0100 1011 0000		
000000	10010	01000	01000	00000	100000
101011	01001	01000	0000 0100 1011 0000		

5 Instruções para tomada de decisões

De uma forma bem geral, uma das características que distingue o computador de uma calculadora é sua habilidade de tomar decisões. Baseado nos dados e valores calculados durante uma computação, diferentes instruções podem ser executadas. MIPS inclui duas instruções de tomada de decisões que são similares ao *if* e ao *goto* presentes na linguagem C. As instruções são mostradas a seguir.

beq register1, register2, L1

A instrução **beq** (*branch if equal*) desvia a execução para o *label* (rótulo) L1 se o valor dos registradores forem iguais. A segunda instrução, similar a primeira, é a seguinte:

bne register1, register2, L1

A instrução **bne** (*branch if not equal*) desvia a execução para o *label* (rótulo) L1 se o valor dos registradores **não** forem iguais.

Os rótulos (L1) mencionados nas instruções são semelhantes aos estabelecidos em C quando a instrução *goto* é utilizada. Em um baixo nível tais rótulos referenciam um endereço na memória, uma vez em que as instruções do programa em execução estão armazenadas na memória. Por isso, os rótulos indicam endereços de memória que contém instruções. Uma vez em que as instruções MIPS possuem sempre 32 bits, elas são referenciadas como qualquer outra palavra na memória. O exemplo a seguir ilustra a utilização das instruções de desvio

Exemplo:

Dado o seguinte trecho de código em C, onde as variáveis f, g, h, i, j correspondem aos registradores \$s0 à \$s4, qual é o código MIPS correspondente?

```
if(i == j)
    goto L1;

f = g + h;

L1: f = f - i;
```

O código MIPS correspondente é o seguinte:

```
beq $s3, $s4, L1

add $s0, $s1, $s2

L1: sub $s0, $s0, $s3
```

Exemplo (if-then-else):

Utilizando a mesma associação entre variáveis e registradores do exemplo anterior, gere as instruções de máquina do seguinte código em C:

```
if(i == j)
    f = g + h;
else
    f = g - h;

MIPS:

bne $s3, $s4, Else

add $s0, $s1, $s2
```

As instruções mostradas representa a parte referente ao if. Com isso, depois de executar a instrução `add` é preciso ir para o fim do if-then-else. Pois se a instrução dentro do if foi executada a instrução referente ao else deve ser ignorada. Para isso, é utilizado um outro tipo de instrução de desvio, chamada de **instrução de desvio incondicional**. A instrução é a **j** (jump) que desvia incondicionalmente a execução do programa para o *label* (rótulo) indicado. A instrução é utilizada no exemplo e é mostrada a seguir.

```
j Exit

Else: sub $s0, $s1, $s2

Exit:
```

O teste de igualdade ou desigualdade é provavelmente o teste mais popular, mas em alguns casos é preciso verificar se uma variável é menor do que outra. Tais comparações são viabilizadas na linguagem de montagem MIPS com uma instrução que compara dois registradores e atribui o valor 1 a um terceiro registrador se o primeiro é menor do que o segundo, caso contrário o valor 0 é atribuído. A instrução comentada é a `slt` (set on less than).

```
slt $t0, $s3, $s4
```

Outra instrução de desvio disponível na arquitetura é a instrução `jr register` que desvia a execução do programa para o endereço contido no registrador `register`. O exemplo a seguir aplica as instruções apresentadas.

Exemplo:

Dado o código em C:

```
while(save[i] == k)
    i = i+j;
```

Assumindo que `i`, `j` e `k` correspondem aos registradores `$s3`, `$s4` e `$s5` e a base do vetor `save` está em `$s6`. Qual é o código MIPS correspondente ao segmento em C apresentado?

O primeiro passo é armazenar o valor `save[i]` em um registrador temporário.

```
Loop: add $t1, $s3, $s3 # registrador temporário $t1 = 2 * i
      add $t1, $t1, $t1 # registrador temporário $t1 = 4 * i
      add $t1, $t1, $s6 # registrador temporário $t1 = endereço de save[i]
```

```
      lw $t0, 0($t1)    # registrador temporário $t0 = save[i]
```

A próxima instrução realiza o teste do loop, saindo se `save[i] ≠ k`:

```
      bne $t0, $s5, Exit # vá para Exit se save[i] é diferente de k$
      add $s3, $s3, $s4
      j Loop
```

```
Exit:
```

Exemplo:

Qual o código para testar se uma variável `a`, correspondendo ao registrador `$s0`, é menor do que a variável `b`, representada pelo registrador `$s1`? Feito o teste, a execução é então desviada para o rótulo `Less` se a condição for verdadeira.

```
slt $t0, $s0, $s1    # $t0 recebe 1 se $s0 < $s1 (a < b)
```

```
bne $t0, $zero, Less # vai para Less se $t0 é diferente de 0, isto é, se a < b
```

Nota: O registrador `$zero` utilizado no exemplo sempre contém o valor 0.

A Tabela 3 a seguir resume as instruções apresentadas na aula, incluindo seus formatos e campos.

Nome	Formato	Exemplo								Comentário
		0	18	19	17	0	32			
add	R	0	18	19	17	0	32			add \$s1, \$s2, \$s3
sub	R	0	18	19	17	0	34			sub \$s1, \$s2, \$s3
lw	I	35	18	17		100				lw \$s1, 100(\$s2)
sw	I	43	18	17		100				sw \$s1, 100(\$s2)
beq	I	4	17	18		25				beq \$s1, \$s2, 100
bne	I	5	17	18		25				bne \$s1, \$s2, 100
slt	R	0	18	19	17	0	42			slt \$s1, \$s2, \$s3
j	J	2	2500					j 10000		
jr	R	0	9	0	0	0	8			jr \$t1
Tamanho		6 bits	5 bits	5 bits	5 bits	5 bits	6 bits			Instruções MIPS de 32 bits
tipo-R	R	op	rs	rt	rd	shamt	funct			Formato instr. aritméticas
tipo-I	I	op	rs	rt	address					Formato transf. de dados e desvios

Tabela 3: Instruções MIPS, seus campos e formatos