

Linguagem de Montagem - Aula 3

Nicolas Chagas Souza

20/06/2022

Relembrando

Na última aula vimos:

- Instruções aritméticas
 - Exemplos: add, sub e mul.
 - * Sintaxe: add \$t0, \$t1, \$t2 # t0 = t1 + t2
- Simulador que utilizaremos: SPIM
- Tabela de registradores (32 registrados na arquitetura MIPS)
- Pseudo instruções
- Estrutura de um programa

System calls (syscalls)

São chamadas a tarefas, feitas a nível lógico pelo processador, executadas por softwares de sistema (sistema operacional, por exemplo).

Como fazer uma syscall:

- Carregar em \$v0 o código da syscall;
- Carregar os argumentos necessários em \$a0 a \$a3;
- Executar a instrução **syscall**;
- Obtém o resultado, se aplicável, em \$v0.

Uma tabela com os códigos das principais syscalls está disponível na página A-44 do 5ª edição do livro texto.

Exemplo: Imprimir o inteiro do registrador \$s0 na tela.

```
.text:
main:
    addi $v0, $zero, 1 # Código 1 imprime um inteiro na tela.
    addi $a0, $s0, 0 # Guardando o valor de s0 em a0.
    syscall # Imprime o inteiro na tela
```

Rótulos

Um programa é um *vetor de instruções*. Cada instrução ocupa uma posição na memória (sequencial). Quando queremos acessar determinada instrução, pelo seu endereço, podemos usar *rótulos*.

Exemplo:

```
...
imprime_inteiro: # Rótulo.
    li $v0, 1
    move $a0, $s0
    syscall
...
```

Estrutura do programa

```
.data:
ola: .asciiz "Ola Mundo\n" #0 processador criará a string na memória e 'ola' é um rótulo para a primeira p
.text:
main:
    li $v0, 4 #Código da syscall.
    la $a0, ola # Passa o endereço da string a ser impressa.
    syscall

    li $v0, 10
    syscall # Encerra o programa (return 0).
```

Acesso à memória

- Cada célula de memória armazena 1 byte (8 bits). O byte é utilizado já que o tamanho de outros tipos de dados é múltiplo deles
- Blocos de 4 bytes (32 bits) são muito utilizados.
- Um dado sempre ocupa um endereço múltiplo de 4 (restrição de alinhamento) para que não ocorra desperdício de espaço.

Instruções de acesso a memória

As instruções load word (lw) e store word (sw) possuem a seguinte sintaxe, sendo **base** um registrador e **offset** uma constante:

- lw **reg_destino**, **offset(base)** # Carrega um dado da memória em **reg_destino**.
- sw **reg_origem**, **offset(base)** # Escreve o conteúdo de **reg_origem** na memória.

Essas instruções acessam o endereço de memória **base + offset**. Utiliza-se a soma da base com o offset para que seja possível acessar um maior número de endereços, já que a base é um registrador (32 bits, em linguagem de máquina) e o offset é uma constante (16 bits).

Observação: o offset é o número de bytes que precisam ser deslocados a partir do endereço base.

Exemplos: Utilizando o seguinte mapeamento g em \$s1, h em \$s2 e o endereço base de A em \$s3.

1. Exemplo 1

C	MIPS
g = h + A[8];	lw \$t0, 32(\$s3) # t0 = A[8] add \$s1, \$s2, \$t0 # g = h + A[8]

2. Exemplo 2

C	MIPS
A[12] = g + A[6];	lw \$t0, 24(\$s3) add \$t0, \$t0, \$s1 sw \$t0, 48(\$s3)