Aula 03 - Syscalls e Registradores

▼ Revisão da aula anterior

```
add $t0, $s0, $s1
```

- add é o minemônico, $t0 sempre vai ser o resultado e os dois próximos valores são sempre os operandos, tem que seguir essa ordem.
- Instruções aritmética sempre operam com registradores.

▼ **Instruções Aritméticas e instruções no Assembly**

- O tamanho de 32 bits (4 bytes) é muito comum na arquitetura MIPS e é chamado de palavra.

▼ Escrevendo um programa em assembly MIPS

- Um programa no assembly MIPS possui a seguinte estrutura:

```
.data
#alocação de memória
.text
main:
#código em assembly
#encerramento do código (não necessariamente por último)
```

- data e text não precisam estar em ordem para funcionar.
- Alocação de memória é quando declaramos variáveis nas linguagens de alto nível.
- .data ⇒ alocação na memória principal (RAM).
- Obs.1: Memória principal é a memória volátil onde dados ficam disponíveis para o processador em complemento aos registradores. Geralmente composta pela Cache e pela RAM.

- Obs.2: Memória secundária é a unidade persistente, onde os dados são armazenados, como disco rígido e pendrive.
- “main:” ponto de partida / rótulo do assembly para indicar o início do código como o main() do C. Também devemos indicar o encerramento com um código.

▼ Declaração de dados na seção .data

- Segue o seguinte formato:
- rótulo: .tipo valor1, valor 2, … , valorN.
- Rótulo é um apelido para o endereço de memória, então é o nome de uma variável.
- Se eu alocar N valores, eu estou alocando um vetor estático (tem um tamanho fixo).
- Os possíveis tipos são:
- word w1, w2, … , wn: representam palavras de 32 bits por w1, w2, … , wn que são inteiros.
- half h1, h2, … , hn: representam dados de 16 bits (2 bytes).
- byte b1, b2, … , bn: representam dados de 8 bits (1 byte).
- asciiz str: representa uma cadeia de caracter dada em str (entre aspas). A cadeia é automaticamente terminada pelo caracter nulo.

Chamadas ao sistema (syscalls)

- São usados para executar tarefas específicas sob responsabilidade de um software de sistema:
  - entrada / saída
  - alocação dinâmica de memória.
- Para executar uma Syscall:

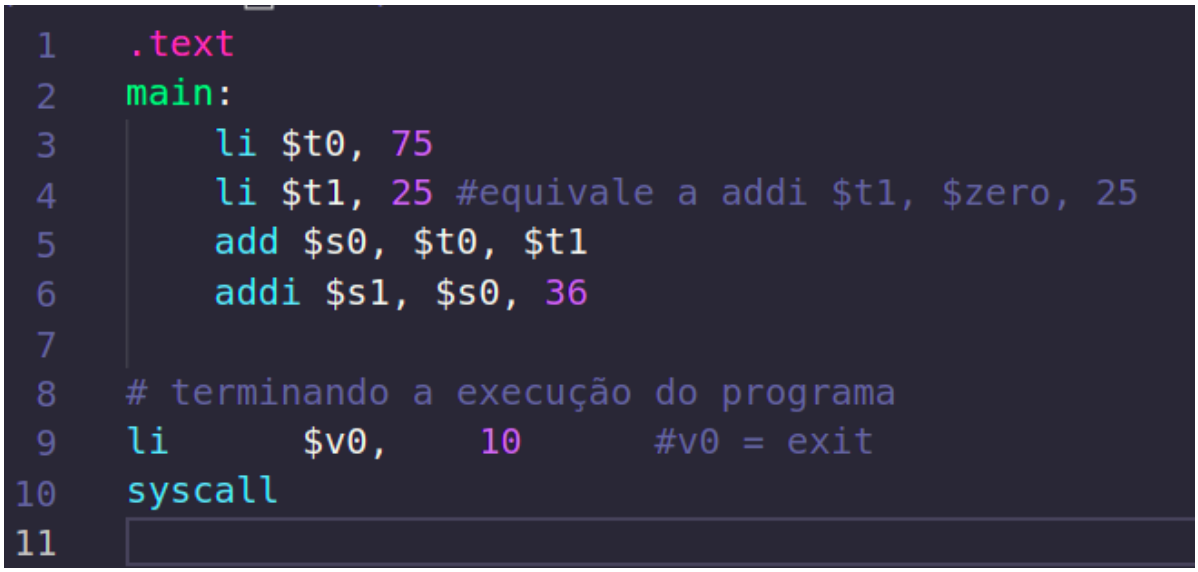
1. carrega-se o código da syscall em $v0.
2. Carrega-se os argumentos da syscall nos registradores $a0 e $a1 (de acordo com a Syscall).

3. Executa com o comando Syscall.
4. Obtém-se o retorno no registrador $v0 (de acordo com a Syscall).

- Obs: uma tabela com os códigos das possíveis Syscalls será disponibilizada na página da disciplina.

Como traduzir uma pseudoinstrução em instrução da Arquitetura ? - Pergunta de Prova

não tenho certeza se esta é a resposta :/



```
1   .text
2   main:
3       li $t0, 75
4       li $t1, 25 #equivale a addi $t1, $zero, 25
5       add $s0, $t0, $t1
6       addi $s1, $s0, 36
7
8   # terminando a execução do programa
9   li      $v0,    10      #v0 = exit
10  syscall
11
```

Pseudoinstruções úteis:

1: li reg, const ⇒ carrega uma constante num registrador.

li = load immediate

Exemplo:

```
li $v0, 1 #$v0 = 1
```

2: la reg, lavel ⇒ carrega o endereço de memória de um rótulo num registrador.

la = load address

Exemplo:

```
.data
ola: .asciiz "Ola Mundo.\n"
.text
la $a0, ola #$a0 recebe o endereço da string ola.
```

3: move reg1, reg 2 ⇒ copia o conteúdo de reg2 para reg1

Exemplo:

```
move $t0, $v0 #$t0 = $v0
```

Exemplo: Para imprimir na tela a string ola declarada acima:

```
li $v0, 4
la $a0, ola
syscall
```

Ex: Encerrar um programa:

```
li $v0, 10
syscall
```