

# Aula 12:

# Subprogramas – MIPS

*Disciplina:* Organização e Arquitetura de Computadores

**Prof. Luiz Olmes**

*olmes@unifei.edu.br*

## Nas aulas anteriores...

---

### ▶ **O QUE JÁ ESTUDAMOS?**

- ▶ Evolução das máquinas.
- ▶ Processador.
- ▶ Memória.
- ▶ Barramentos.
- ▶ Introdução à Linguagem de Montagem.
- ▶ Condicionais, diretivas, syscalls.
- ▶ Memória e arrays.
- ▶ Loops.

### ▶ **OBJETIVOS:**

- ▶ Subprogramas:
  - ▶ Fluxo de controle
  - ▶ Fluxo de dados
  - ▶ Subprogramas

# Fluxo de controle

---

- ▶ A invocação de funções em linguagem de alto nível **desvia** o fluxo de execução de um programa **duas** vezes:
  - ▶ Quando a função é **chamada**.
  - ▶ Quando se **retorna** da função.

```
1. int main()  
2. {  
3.     ...  
4.     a = fatorial(2);  
5.     b = fatorial(4);  
6.     ...  
7. }  
8.
```

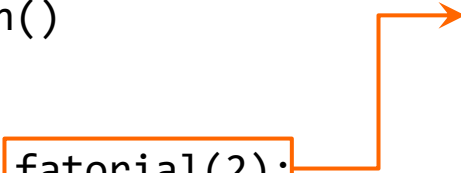
```
9. int fatorial(int n)  
10. {  
11.     int i, f = 1;  
12.  
13.     for(i = n; i > 1; i--)  
14.         f = f * i;  
15.  
16.     return f;  
17. }
```

# Fluxo de controle

---

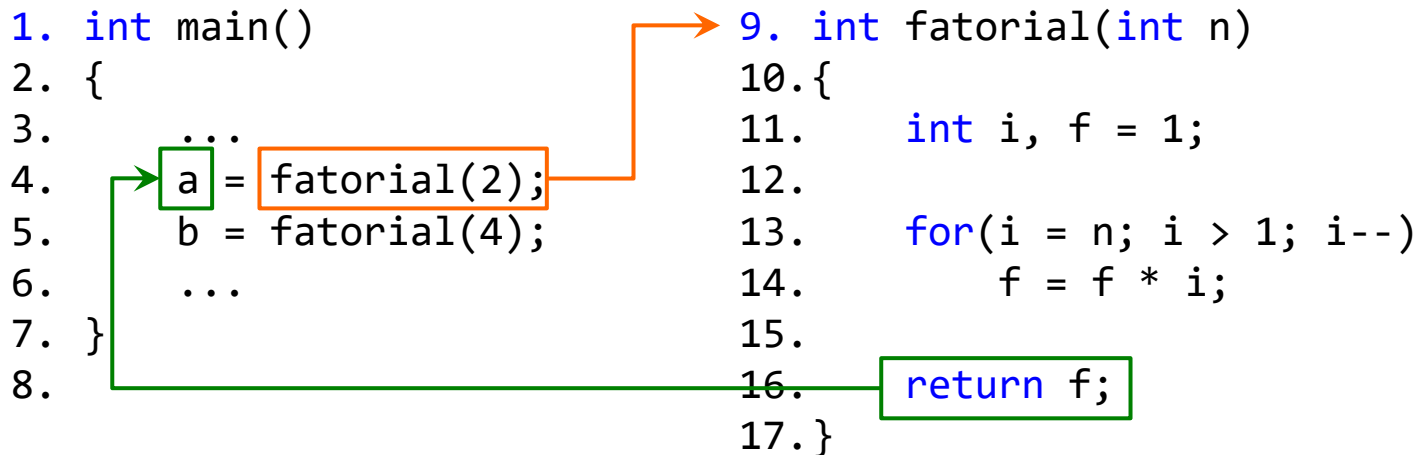
- ▶ A invocação de funções em linguagem de alto nível **desvia** o fluxo de execução de um programa **duas** vezes:
  - ▶ Quando a função é **chamada**.
  - ▶ Quando se **retorna** da função.

```
1. int main()  
2. {  
3.     ...  
4.     a = fatorial(2);  
5.     b = fatorial(4);  
6.     ...  
7. }  
8.  
  
9. int fatorial(int n)  
10. {  
11.     int i, f = 1;  
12.  
13.     for(i = n; i > 1; i--)  
14.         f = f * i;  
15.  
16.     return f;  
17. }
```

An orange arrow originates from the call to the `fatorial(2)` function in line 4 of the `main()` function and points to the definition of the `fatorial` function starting at line 9. Additionally, the `fatorial(2);` expression in line 4 is enclosed in an orange rectangular box.

# Fluxo de controle

- ▶ A invocação de funções em linguagem de alto nível **desvia** o fluxo de execução de um programa **duas** vezes:
  - ▶ Quando a função é **chamada**.
  - ▶ Quando se **retorna** da função.



# Fluxo de controle

---

- ▶ Neste exemplo, a função `main()` invoca a função `fatorial()` duas vezes, e `fatorial()` **retorna duas vezes**, porém, para **locais diferentes** na `main()`.
- ▶ A cada chamada de `fatorial()`, a CPU precisa se lembrar do **endereço de retorno** apropriado na `main()`.
- ▶ Nota-se, ainda, que a `main()` **também é uma função**. Ela é invocada pelo Sistema Operacional quando o programa é executado.

```
1. int main()
2. {
3.     ...
4.     a = fatorial(2);
5.     b = fatorial(4);
6.     ...
7. }
8.
9. int fatorial(int n)
10. {
11.     int i, f = 1;
12.
13.     for(i = n; i > 1; i--)
14.         f = f * i;
15.
16.     return f;
17. }
```

# Subprogramas: representação de funções

---

- ▶ A representação de funções em linguagem de montagem do MIPS é realizada através de **labels**, seções de **texto** (.text) e **dados** (.data).

```
1. int main()
2. {
3.     ...
4. }
5.
6.
7. int fatorial(int n)
8. {
9.     ...
10. }
```

```
1. # main
2. .data
3.     ...
4. .text
5. main:
6.     ...
7.
8. # fatorial
9. .data
10.    ...
11. .text
12. fatorial:
13.    ...
```

## Subprogramas: representação de funções

---

- ▶ A representação de funções em linguagem de montagem do MIPS é realizada através de **labels**, seções de **texto** (.text) e **dados** (.data).
- ▶ Em assembly, o equivalente às funções das linguagens de alto nível são os **subprogramas**.
- ▶ A ideia de um subprograma é reunir um conjunto de instruções escritas em assembly de forma que elas possam ser executadas sempre que necessário.
- ▶ Normalmente, estas instruções estão associadas a tarefas rotineiras. Dessa forma, subprogramas permitem **reutilizar** suas instruções sem a necessidade de reprogramá-las, o que pode levar a erros.



# Fluxo de controle no MIPS

---

- ▶ No MIPS, a instrução **jal** (Jump And Link) é utilizada para invocar funções.
  - ▶ A instrução **jal** salva o endereço de retorno (o endereço da próxima instrução: PC) em um registrador especialmente dedicado para esse fim (**\$ra**), antes de saltar para o subprograma.
  - ▶ **jal** é a **única** instrução do MIPS capaz de acessar o valor do **Program Counter**, de modo que ela pode armazenar o valor de  $PC + 4$  em **\$ra**.
- ▶ Para retornar à função chamadora, é necessário transferir o controle novamente, de modo a saltar para o endereço armazenado em **\$ra**. Isso pode ser feito através da instrução **jr** (Jump Register):
  - ▶ **jr \$ra**
- ▶ Ambas as instruções **jal** e **jr** são do **tipo J**.

# Fluxo de dados

---

- ▶ Nas linguagens de alto nível, é comum que funções **recebam parâmetros** e produzam **valores de retorno**.
- ▶ No código de exemplo, as partes **vermelhas** do programa mostram os parâmetros formais da função `fatorial()`.
- ▶ Já os destaques em **azul** especificam os valores de retorno.

```
1. int main()  
2. {  
3.     ...  
4.     a = fatorial(2);  
5.     b = fatorial(4);  
6.     ...  
7. }  
8.  
9. int fatorial(int n)  
10. {  
11.     int i, f = 1;  
12.  
13.     for(i = n; i > 1; i--)  
14.         f = f * i;  
15.  
16.     return f;  
17. }
```

# Fluxo de dados no MIPS

---

- ▶ O MIPS emprega as seguintes convenções para argumentos de subprogramas e valores de retorno:
- ▶ Um máximo de 4 valores podem ser repassados como argumentos aos subprogramas. Para isso, eles devem ser colocados nos registradores `$a0` a `$a3` antes da chamada da instrução `jal`.
  - ▶ Mais que 4 argumentos necessitam de usar a pilha (stack).
- ▶ Um subprograma pode devolver um máximo de 2 valores. Estes valores de retorno devem ser colocados nos registradores `$v0` ou `$v1` antes da chamada da instrução `jr`.
- ▶ Estas convenções não são forçadas pelo hardware MIPS ou pelo assembler. Entretanto, os desenvolvedores devem segui-las para que subprogramas escritos por outros desenvolvedores possam ser compatíveis entre si.

# Tipos de dados

---

- ▶ A linguagem assembly **não é tipada**.
- ▶ Isso significa que a linguagem **não faz distinção** entre inteiros, caracteres, ponteiros e outros tipos existentes nas linguagens de alto nível.
- ▶ Dessa forma, é responsabilidade do **programador** assembly **realizar a verificação de tipos** de seu programa.
- ▶ Em particular, o programador deve garantir que os argumentos dos subprogramas e os valores de retorno sejam usados de forma consistente.
  - ▶ **Exemplo:** o que aconteceria se o endereço de um valor (ao invés do valor) fosse passado como argumento da função `fatorial()`?

# Subprogramas

---

- ▶ **Exemplo 1:** escrever, em MIPS assembly, a função `fatorial()`, como mostrada no slide 3.
- ▶ **Exemplo 2:** escrever, em MIPS assembly, uma função que verifique se um número é um quadrado perfeito.
- ▶ **Exemplo 3:** escrever, em MIPS assembly, uma função que verifique se um número é par.

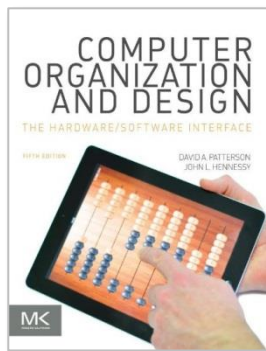
# Dúvidas?

---



# Sugestão de Estudo

- ▶ **Capítulo 2 e Apêndice A:**  
PATTERSON, D. A.; HENESSY, J. L.  
*Computer Organization and Design*. 2013.



- ▶ **Capítulo 7:** TANENBAUM, A. S.; TODD, A. *Organização Estruturada de Computadores*, 2007



# Aula 12:

# Subprogramas – MIPS

*Disciplina: Organização e Arquitetura de Computadores*

**Prof. Luiz Olmes**

*olmes@unifei.edu.br*