**Universidade Federal de Pelotas**
**Instituto de Física e Matemática**
Departamento de Informática
**Bacharelado em Ciência da Computação**

# Arquitetura e Organização de Computadores I

## Aula 28

### Arquitetura do Processador MIPS: conjunto de instruções e programação em linguagem simbólica

**Prof. José Luís Güntzel**

**guntzel@ufpel.edu.br**

**www.ufpel.edu.br/~guntzel/AOC1/AOC1.html**

# Arquitetura do MIPS

## Suporte de Hardware para Procedimentos

**Motivos para o uso de procedimentos:**

- **Tornar o programa mais fácil de ser entendido**
- **Permitir a reutilização do código do procedimento**
- **o conceito de procedimento permite que o programador se concentre em uma parte do código (os parâmeros funcionam como barrreira)**

# Arquitetura do MIPS

## Suporte de Hardware para Procedimentos

**Passos que o programa e o procedimento precisam executar:**

1. **Colocar os parâmetros em um lugar onde eles possam ser acessados pelo procedimento**
2. **Transferir o controle para um procedimento**
3. **Garantir os recursos de memória necessários à execução do procedimento**
4. **Realizar a tarefa desejada**
5. **Colocar o resultado em um lugar acessível ao programa que chamou o procedimento**
6. **Retornar o controle para o ponto de origem**

# Arquitetura do MIPS

## Suporte de Hardware para Procedimentos

**O Software do MIPS utiliza os seguintes registradores na implementação de chamada de procedimento:**

- $a0-$a3: **quatro registradores para argumento, através dos quais são passados parâmetros do programa para o procedimento**

**Registradores adicionais são implementados na pilha (que fica na memória), usando o apontador da pilha:** $sp

- $v0-$v1: **dois registradores para retorno de valores do procedimento para o programa**
- $ra: **um registrador que contém o endereço para o procedimento retornar ao ponto de origem**

# Arquitetura do MIPS

## Suporte de Hardware para Procedimentos

**Instrução *jump and link*:** jal endereço-do-procedimento

- **Desvia para um endereço e ao mesmo tempo salva o endereço da instrução seguinte (PC+4) no registrador** $ra

**Instrução *jump register*:** jr $ra

- **Desvia para um endereço armazenado no registrador** $ra

# Arquitetura do MIPS

## Suporte de Hardware para Procedimentos

**Resumindo a execução de um procedimento no MIPS:**

1. **O programa chamador coloca os valores dos parâmetros em** $a0-$a3
2. **O programa chamador usa** jal X **para desviar para o procedimento que está em** X
3. **O procedimento executa**
4. **O procedimento executa armazena os resultados em** $v0-$v1
5. **O procedimento retorna o controle para o programa chamador usando** jr $ra

# Arquitetura do MIPS

## Suporte de Hardware para Procedimentos

### Compilação de um Procedimento-Folha
### (i.e.: um procedimento que não chama outro procedimento)

```
int leaf_example (int g, int h, int i, int j )
{
        int f;
        f = (g + h) - (i + j);
        return f;
}
```

- **Vamos supor que possamos somar ou subtrair valores como 4, 8 ou 12 ao conteúdo de um dado registrador**

# Suporte de Hardware para Procedimentos

## Compilação de um Procedimento-Folha

- g, h, i, j **passadas como parâmetros em** $a0, $a1, $a2, $a3
- **Resultado em** $s0, **retorna por** $v0
- **O código gerado pelo compilador inicia por**

```
leaf_example
```

# Arquitetura do MIPS

## Suporte de Hardware para Procedimentos

### Compilação de um Procedimento-Folha

- **Partindo do pressuposte que os valores antigos de** $s0, $t0 **e** $t1 **precisam ser mantidos intactos…**
- **Primeiro, é necessário salvar na pilha os registradores** $s0, $t0 **e** $t1**, para que possam ser usados:**

```
sub   $sp, $sp, 12     # ajusta a pilha para abrir espaço para guardar 3 itens
sw    $t1, 8($sp)      # salva o conteúdo do registrador $t1 para preservá-lo
sw    $t0, 4($sp)      # salva o conteúdo do registrador $t0 para preservá-lo
sw    $s0, 0($sp)      # salva o conteúdo do registrador $s0 para preservá-lo
```

# Arquitetura do MIPS

## Suporte de Hardware para Procedimentos

### Compilação de um Procedimento-Folha

- **Corpo do procedimento, idêntico ao exemplo da aula passada**

```
add   $t0, $a0, $a1      # registrador $t0 contém g + h
add   $t1, $a2, $a3      # registrador $t1 contém i + j
sub   $s0, $t0, $t1      # f recebe $t0 - $t1, resultado final
```

- **O valor de retorno, a ser armazenado em** f**, será copiado em um dos registradores de retorno (**$v0 **ou** $v1**)**

```
add   $v0, $s0, $zero    # retorna f ( $v0 □  $s0 + 0 )
```

# Suporte de Hardware para Procedimentos

## Compilação de um Procedimento-Folha

- **Restaurando os valores do registradores que haviam sido empilhados**

```
lw    $s0, 0($sp)      # restaura o valor de $s0 para o chamador
lw    $t0, 4($sp)      # restaura o valor de $t0 para o chamador
lw    $t1, 8($sp)      # restaura o valor de $t1 para o chamador
add   $sp, $sp, 12     # ajusta a pilha de modo a remover 3 itens
```

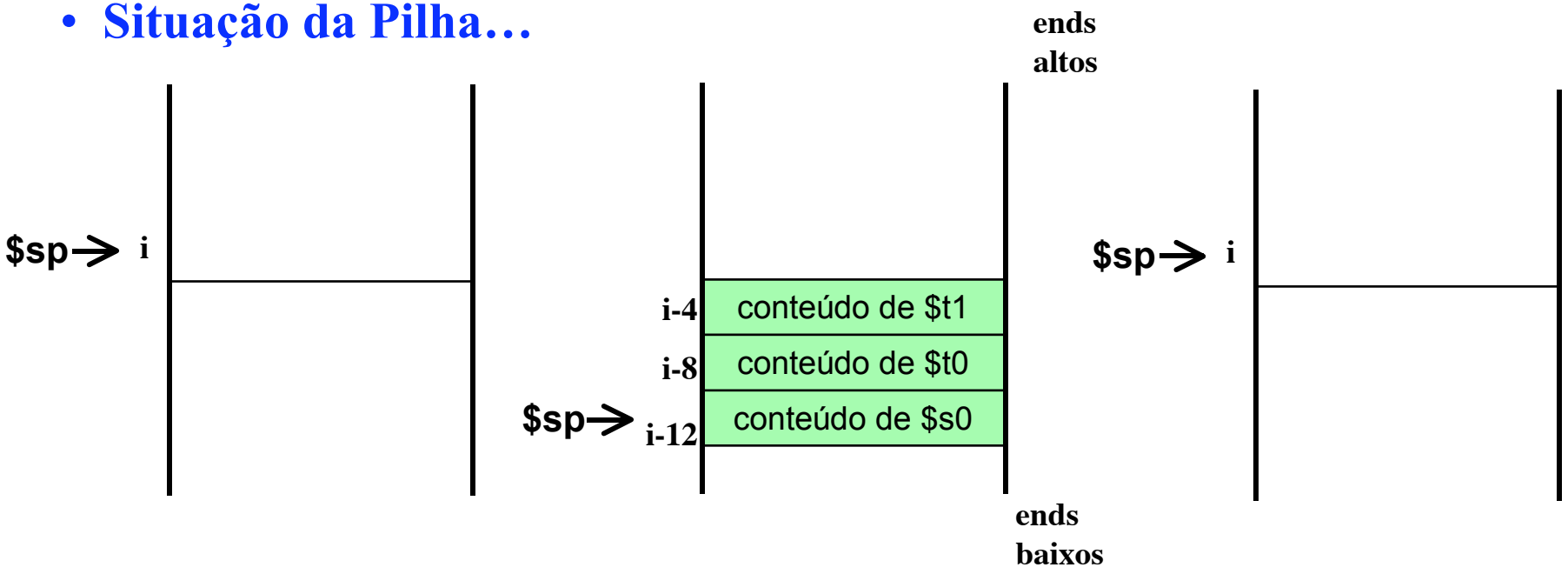
- **Conclusão do procedimento, desvio para o endereço de retorno:**

```
jr    $ra              # desvia de volta para o programa que chamou
```

# Suporte de Hardware para Procedimentos

## Compilação de um Procedimento-Folha

- Situação da Pilha…



# Arquitetura do MIPS

## Suporte de Hardware para Procedimentos

### Compilação de um Procedimento-Folha

**Para evitar o salvamento/recuperação de regs. Temporários, o software do MIPS disponibiliza duas classes de regs.:**

- $t0-$t9: **dez registradores temporários que não são preservados pelo procedimento chamado**
- $s0-$s7: **oito registradores de salvamento (se usados, o procedimento chamado precisa salvar seus valores e depois restaurá-los)**

# Suporte de Hardware para Procedimentos

## Compilação de um Procedimento-Folha

- **No exemplo anterior, as seguintes instruções podem ser eliminadas:**

```
sw    $t1, 8($sp)     # salva o conteúdo do registrador $t1 para preservá-lo
sw    $t0, 4($sp)     # salva o conteúdo do registrador $t0 para preservá-lo
```

```
lw    $t0, 4($sp)     # restaura o valor de $t0 para o chamador
lw    $t1, 8($sp)     # restaura o valor de $t1 para o chamador
```

# Arquitetura do MIPS

## Suporte de Hardware para Procedimentos

### Compilação de um Procedimentos Aninhados
### (i.e.: um procedimento que chama outro procedimento)

- O procedimento chamador coloca na pilha todos os registradores de argumento ($a0-$a3) ou registradores temporários ($t0-$t9) que sejam necessário após a chamada
- O procedimento chamado coloca na pilha o endereço de retorno (armazenado em $ra) e todos os registradores de salvmento usados por ele ($s0-$s7)
- O apontadorda pilha ($sp) é ajustado para acomodar a quantidade de registradores colocados na pilha
- Quando do retorno, os valores dos registradores são restaurados a partir da pilha e $sp é atualizado

# Suporte de Hardware para Procedimentos

## Compilação de um Procedimentos Aninhados

**Seja a seguinte função escrita em C:**

```
int fact(int n )
{
        if (n<1)  return (1);
        else  return (n* fact(n-1));
}
```

- **Suponha que se pode somar ou subtrair constantes 1 ou 4 ao conteúdo dos registradores**

# Suporte de Hardware para Procedimentos

## Compilação de um Procedimentos Aninhados

- Parâmetro passado pela variável n corresponde a **$a0**
- O programa compilado inicia com o label do procedimento
- Depois salva **$ra** e **$a0** na pilha

fact:

```
sub   $sp, $sp, 8      # ajusta a pilha para abrir espaço para receber 2 itens
sw    $ra, 4($sp)      # salva o endereço de retorno
sw    $a0, 0($sp)      # salva o argumento n
```

# Suporte de Hardware para Procedimentos

## Compilação de um Procedimentos Aninhados

- **Da primeira vez que** fact **for chamado, a instrução** sw **salva um endereço no programa que chamou** fact
- **As duas instruções seguintes testam se** n **é menor que 1, desviando para** L1 **se** n >=1

```
slt   $t0, $a0, 1      # testa se n < 1
beq   $t0, $zero, L1   # se n >= 1, desvia para L1
```

# Arquitetura do MIPS

## Suporte de Hardware para Procedimentos

### Compilação de um Procedimentos Aninhados

- **Se** n <1**, fact retorna o valor 1, colocando 1 no registrador de valor (soma 1 com 0 e coloca o resultado em** $v0**)**
- **Depois, retira dois valores de registradores de salvamento da pilha e desvia para o endereço de retorno**

```
add   $v0, $zero, 1     # retorna o valor 1
add   $sp, $sp, 8       # elimina 2 itens da pilha
jr    $ra               # retorna para depois da instrução jal
```

# Suporte de Hardware para Procedimentos

## Compilação de um Procedimentos Aninhados

- **Antes de retirar os dois valores da pilha, devemos verificar se não houve necessidade de carregar** $a0 **e** $ra

```
L1:  sub   $a0, $a0, 1     # n>=1, o argumento recebe (n-1)
     jal   fact            # chama fact com argumento (n-1)
```

# Suporte de Hardware para Procedimentos

## Compilação de um Procedimentos Aninhados

- **A próxima instrução é onde** fact **retorna**
- **O endereço de retorno e os argumentos antigos são restaurados, junto com o ponteiro para a pilha (**$sp**)**

```
lw    $a0, 0($sp)     # retorna de jal: restaura argumento n
lw    $ra, 4($sp)     # restaura o endereço de retorno
add   $sp, $sp, 8     # ajusta $sp para eliminar 2 itens
```

# Arquitetura do MIPS

## Suporte de Hardware para Procedimentos

### Compilação de um Procedimentos Aninhados

- Agora, o registrador de valor $v0 recebe o produto do argumento antigo, $a0, e o valor corrente do registrador de valor
- Supondo que tenhamos disponível a instrução mult

```
mult    $v0, $a0, $v0          # retorna n*fact(n-1)
```

- Finalmente, fact desvia novamente para o endereço de retorno:

```
jr      $ra                    # retorna para o chamador
```

## Suporte de Hardware para Procedimentos

### Características dos Registradores usados em Procedimentos

| Não preservados pelo procedimento chamado | Se usados, o procedimento chamado precisa salvar seus valores e depois restaurá-los |
|---|---|
| Reg de salvamento: $s0-$s7 | Reg temporários: $t0-$t9 |
| Reg stack pointer: $sp | Reg de argumento: $a0-$a3 |
| Reg de endereço de retorno: $ra | Reg de retorno de valores: $v0-$v1 |
| Pilha acima do stack pointer | Pilha abaixo do stack pointer |

# Arquitetura do MIPS

## Operandos Imediatos (Constantes)

- **O uso de constantes é muito comum em diversas operações freqüentes (p. ex., incremento, controle de laço etc)**
- **No código do gcc 52% das operações envolvem constantes**
- **No simulador elétrico Spice, 69% das operações envolvem constantes**
- **Com as instruções vistas até aqui, seria preciso buscarmos uma constante na memória:**

```
lw    $t0, addrConstant4($zero)   # $t0 □  4 (constante 4)
add   $sp, $sp, $t0               # $sp □  $sp + 4
```

# Arquitetura do MIPS

## Operandos Imediatos (Constantes)

- **Alternativa para reduzir número de acessos à memória: oferecer versões de instruções aritméticas nas quais um dos operandos é uma constante**
- **Restrição: a constante é mantida dentro da própria instrução**
- **Esta classe de instruções usa o formato I (mesmo de** lw, sw, beq **e** bne**)**
- **Exemplo:**

```
addi    $sp, $sp, 4                 # $sp □  $sp + 4
```

- **O campo reservado para a constante tem tamanho 16 bits**
- **O opcode desta instrução é 8**

# Arquitetura do MIPS

## Operandos Imediatos (Constantes)

- **Operandos imediatos também são muito usados em comparações**
- **Para fazer comparações com valores não zero, existe uma versão imediata da instrução** slt **(**slti**):**

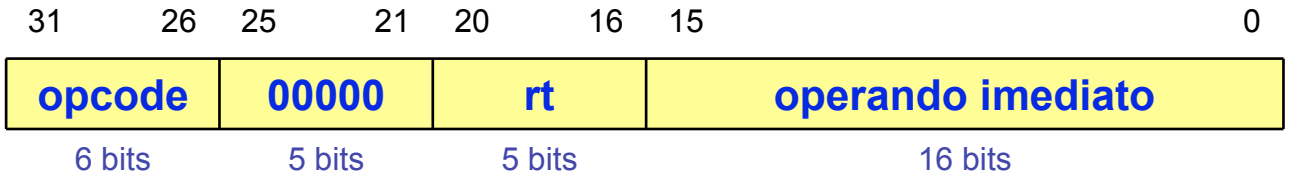
```
slti    $t0, $s2, 10        # se $s2 < 10, então $t0 □ 1
```

# Arquitetura do MIPS

## Operandos Imediatos (Constantes)

**Carga de uma constante de 32 bits em um Registrador**

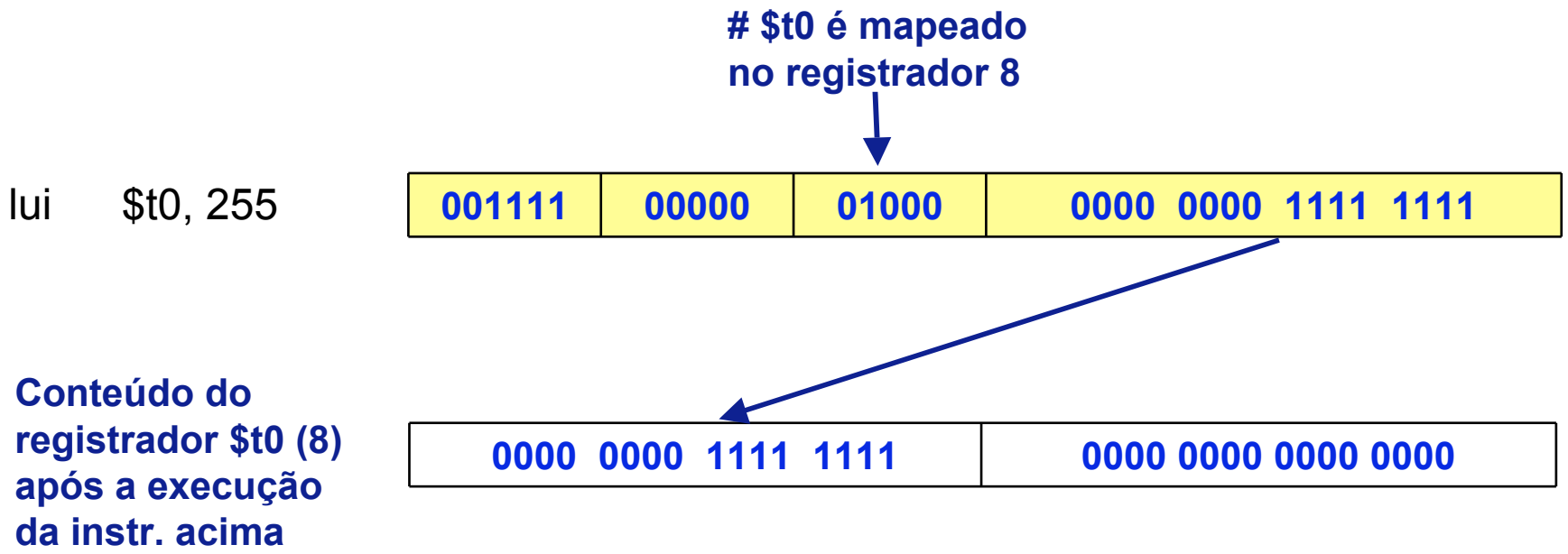
- ***Load upper immediate*** (lui)
- **A instrução** lui **transfere os 16 bits do campo da constante imediata para os 16 mais significativos do registrador especificado,**
- **Os bits menos significativos são preenchido com zero**
- **Esta instrução equivale a multiplicar a constante por $2^{16}$, antes de carregá-la no registrador**



# Arquitetura do MIPS

## Operandos Imediatos (Constantes)

**Carga de uma constante de 32 bits em um Registrador**



# Arquitetura do MIPS

## Operandos Imediatos (Constantes)

**Carga de uma constante de 32 bits em um Registrador**

**Exemplo: qual é o código na linguagem de montagem do MIPS para se carregar a constante de 32 bist abaixo no registrador $s0?**

| 0000 0000 0011 1101 | 0000 1001 0000 0000 |
|---|---|

**1. Carregar os 16 bits mais significativos no registrador, usando a constante 61**

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

**Após a execução desta instrução, o registrador $s0 contém:**

| 0000 0000 0011 1101 | 0000 0000 0000 0000 |
|---|---|

# Arquitetura do MIPS

## Operandos Imediatos (Constantes)

**Carga de uma constante de 32 bits em um Registrador**

**2. Adicionar ao conteúdo do registrador o valor 2.304, expresso em decimal**

```
addi    $s0, $s0, 2304        # 2304 decimal = 0000 1001 0000 0000 binário
```

**Após a execução desta segunda instrução, o registrador $s0 contém:**

| | |
|---|---|
| **0000 0000 0011 1101** | **0000 1001 0000 0000** |