

Arquitetura de Computadores

Linguagem de máquina (Continuação)

Prof. Tiago Gonçalves Botelho

Suporte a procedimentos pelo hardware da máquina

- ❑ A ideia de usarmos procedimento em programas se justifica:
 - ❑ facilidade de entendimento de código
 - ❑ reutilização de código
 - ❑ divisão do problema em problemas menores (divisão e conquista)
 - ❑ Durante a execução de um procedimento, o programa que o chamou e o próprio procedimento, precisam executar seis passos:
 - ❑ Colocar parâmetros em local acessível ao procedimento
 - ❑ Transferir o controle para o procedimento
 - ❑ Garantir recursos de memória para sua execução
 - ❑ Realizar a tarefa
 - ❑ Colocar resultado em local acessível ao programa
 - ❑ Retornar ao ponto de origem

Suporte a procedimentos pelo hardware da máquina

- ❑ Os registradores usados nos procedimentos são:
 - ❑ `$a0 -> $a3` – argumentos para passagem de parâmetros
 - ❑ `$v0 -> $v1` – retorno de valores do procedimento para o programa
- ❑ Ao final de uma chamada ao procedimento é necessário retornar ao ponto de origem.
 - ❑ O MIPS utiliza um registrador que armazena esse ponto de chamada ao procedimento: `$ra (return address)`
 - ❑ O valor do `PC -> $ra`
- ❑ O programa chama o procedimento passando os parâmetros armazenados em `$a0 -> $a3` e usa a função *Jump and Link* para desviar para o procedimento X
 - ❑ `jal X`
- ❑ Os resultados do processamento são armazenados em `$v0 -> $v1` e retorna ao programa usando a função de desvio (*Jump Register*):
 - ❑ `jr $ra`

Banco de registradores e a chamada de procedimentos

Name	Register Number	Usage	Preserve on call?
\$zero	0	constant 0 (hardware)	n.a.
\$at	1	reserved for assembler	n.a.
\$v0 - \$v1	2-3	returned values	no
\$a0 - \$a3	4-7	arguments	yes
\$t0 - \$t7	8-15	temporaries	no
\$s0 - \$s7	16-23	saved values	yes
\$t8 - \$t9	24-25	temporaries	no
\$gp	28	global pointer	yes
\$sp	29	stack pointer	yes
\$fp	30	frame pointer	yes
\$ra	31	return addr (hardware)	yes

Suporte a procedimentos pelo hardware da máquina

- ❑ Se o procedimento precisar de mais registradores para passagem ou retorno de parâmetros, ele deverá usar uma estrutura de pilha do tipo LIFO (*Last In First Out*).
- ❑ O procedimento que precisar usar a pilha deverá apontar para o endereço do topo da pilha, conhecido como *Stack Pointer*.
- ❑ As operações na pilha são:
 - ❑ Push – coloca dados na pilha;
 - ❑ Pop – retira dados da pilha

Suporte a procedimentos pelo hardware da máquina

- ❑ O MIPS reserva um registrador para o *Stack Pointer* $\$sp$
- ❑ A pilha cresce dos endereços mais altos para os mais baixos, assim ao usar o Push o valor do *Stack Pointer* é *decrementado*
- ❑ Lembrar dos comandos `malloc()` e `free()`

High address



Low address

a.

b.

c.

Procedimentos aninhados

- ❑ Exemplo: Seja o seguinte código escrito em linguagem C:

```
int leaf_example (int g, int h, int i, int j)
{
    int f;

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

- ❑ Considere que as variáveis para passagem de parâmetros g, h, i e j são \$a0 -> \$a3 e que f corresponde a \$s0
- ❑ Qual é o código escrito em assembly para o MIPS?

Procedimentos aninhados

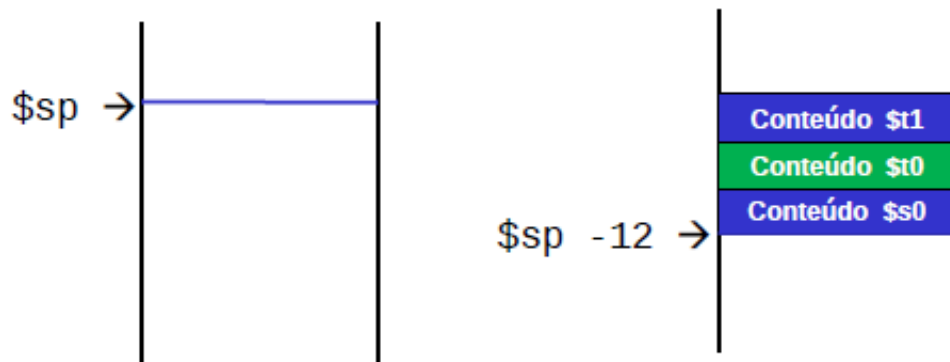
❑ leaf_example:

- ❑ Devemos salvar o conteúdo dos registradores usados pelo procedimento na pilha

```
addi $sp,$sp,-12 # adjust stack to make room for 3 items
sw   $t1, 8($sp) # save register $t1 for use afterwards
sw   $t0, 4($sp) # save register $t0 for use afterwards
sw   $s0, 0($sp) # save register $s0 for use afterwards
```

- ❑ Mais tarde devemos restaurar os valores dos registradores \$t1, \$t0 e \$s0

Endereço
mais alto



Procedimentos aninhados

❑ Corpo do procedimento

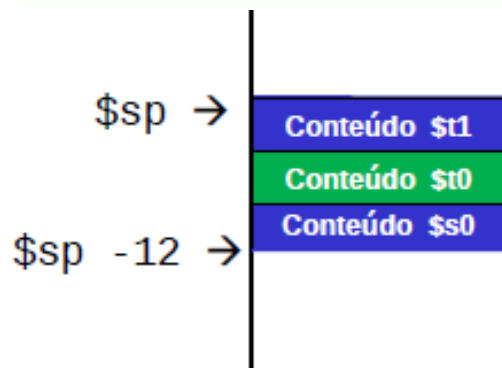
```
add $t0,$a0,$a1 # register $t0 contains g + h
add $t1,$a2,$a3 # register $t1 contains i + j
sub $s0,$t0,$t1 # f = $t0 - $t1, which is (g + h)-(i + j)
```

❑ Retorno do procedimento

```
add $v0,$s0,$zero # returns f ($v0 = $s0 + 0)
```

❑ Restaurando a pilha

```
lw  $s0, 0($sp) # restore register $s0 for caller
lw  $t0, 4($sp) # restore register $t0 for caller
lw  $t1, 8($sp) # restore register $t1 for caller
addi $sp,$sp,12 # adjust stack to delete 3 items
```



Endereço de Retorno:

```
jr $ra # jump back to calling routine
```

Procedimentos aninhados (mais um exemplo)

- Exemplo: Seja o seguinte código escrito em linguagem C:

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

What is the MIPS assembly code?

fact:

```
    addi    $sp,$sp,-8    # adjust stack for 2 items
    sw      $ra, 4($sp)   # save the return address
    sw      $a0, 0($sp)   # save the argument n
```

```
    slti    $t0,$a0,1     # test for  $n < 1$ 
```

```
    beq     $t0,$zero,L1  # if  $n \geq 1$ , go to L1
```

```
    addi    $v0,$zero,1    # return 1
```

```
    addi    $sp,$sp,8      # pop 2 items off stack
```

```
    jr      $ra            # return to after jal
```

```
L1: addi    $a0,$a0,-1     #  $n \geq 1$ : argument gets  $(n - 1)$ 
     jal     fact          # call fact with  $(n - 1)$ 
```

```
    lw      $a0, 0($sp)    # return from jal: restore argument n
```

```
    lw      $ra, 4($sp)    # restore the return address
```

```
    addi    $sp, $sp,8     # adjust stack pointer to pop 2 items
```

```
    mul     $v0,$a0,$v0    # return  $n * \text{fact}(n - 1)$ 
```

```
    jr      $ra            # return to the caller
```

Procedimentos aninhados

- Calculando o fatorial quando $n=3$

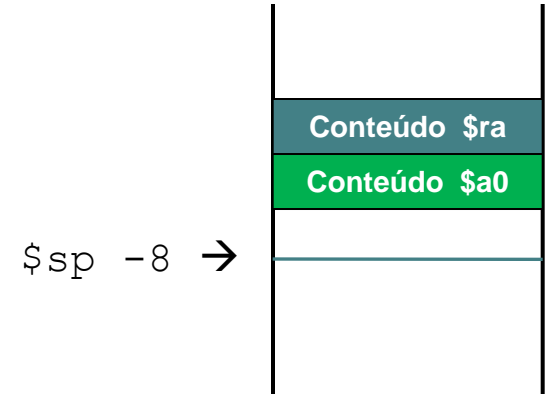
fact:

```
subi $sp,$sp,8
sw  $ra,4($sp)
sw  $a0,0($sp)
```

```
slt $t0,$a0,1    # t0 ← 0; ao ← 3
beq $t0,$zero,L1# desvia
```

L1:

```
subi $a0,$a0,1    # ao ← 2
jal fact          # fact (a0)
```



Procedimentos aninhados

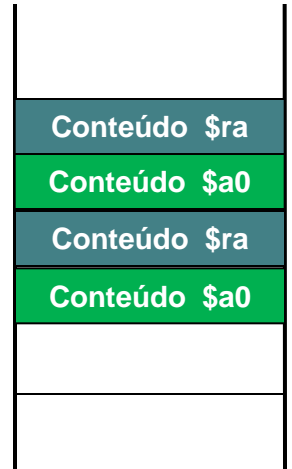
- Calculando o fatorial quando $n=2$

fact:

```
sub $sp, $sp, 8
sw  $ra, 4($sp)
sw  $a0, 0($sp)
```

\$sp -8 →

\$sp -8 →



```
slt $t0, $a0, 1    # t0 ← 0; ao ← 2
beq $t0, $zero, L1 # desvia
```

L1:

```
sub $a0, $a0, 1    # ao ← 1
jal fact           # fact (a0)
```

Procedimentos aninhados

- Calculando o fatorial quando $n=1$

fact:

```
sub $sp,$sp,8
sw  $ra,4($sp)
sw  $a0,0($sp)
```

```
slt $t0,$a0,1    #  $t0 \leftarrow 0$ ;  $ao \leftarrow 1$ 
beq $t0,$zero,L1# desvia
```

$\$sp - 8 \rightarrow$

$\$sp - 8 \rightarrow$

Conteúdo \$ra

Conteúdo \$a0

Conteúdo \$ra

Conteúdo \$a0

Conteúdo \$ra

Conteúdo \$a0

L1:

```
sub $a0,$a0,1    #  $ao \leftarrow 0$ 
jal fact          # fact (a0)
```


Procedimentos aninhados

- Calculando o fatorial quando $n=0$

fact:

```
sub $sp,$sp,8
sw  $ra,4($sp)
sw  $a0,0($sp)
```

```
slt $t0,$a0,1    # t0 ← 1; ao ← 0
beq $t0,$zero,L1 # não desvia
```



```
add $v0,$zero,1    # vo ← 1
lw  $a0,0($sp)     # a0 ← 1
lw  $ra,4($sp)     # endereço da iteração anterior
add $sp,$sp,8      # sobe a pilha
mult $v0,$a0,$v0   # a0 ← 1*1
jr  $ra            # retorna para a iteração anterior
```

\$sp -8 →

\$sp -8 →

Conteúdo \$ra
Conteúdo \$a0
Conteúdo \$ra
Conteúdo \$a0
Conteúdo \$ra
Conteúdo \$a0
Conteúdo \$ra
Conteúdo \$a0

Exemplos “reais”

- PowerPC criado pela IBM e Motorola → Apple Macintosh
 - Semelhanças com o MIPS:
 - 32 registradores para inteiros
 - Instruções de 32 bits
 - Troca de dados com a memória a partir de instruções de load e store.
 - Diferença com o MIPS: possui dois outros modos de endereçamento:
 - Endereçamento indexado: permite que dois registradores sejam referenciados juntos

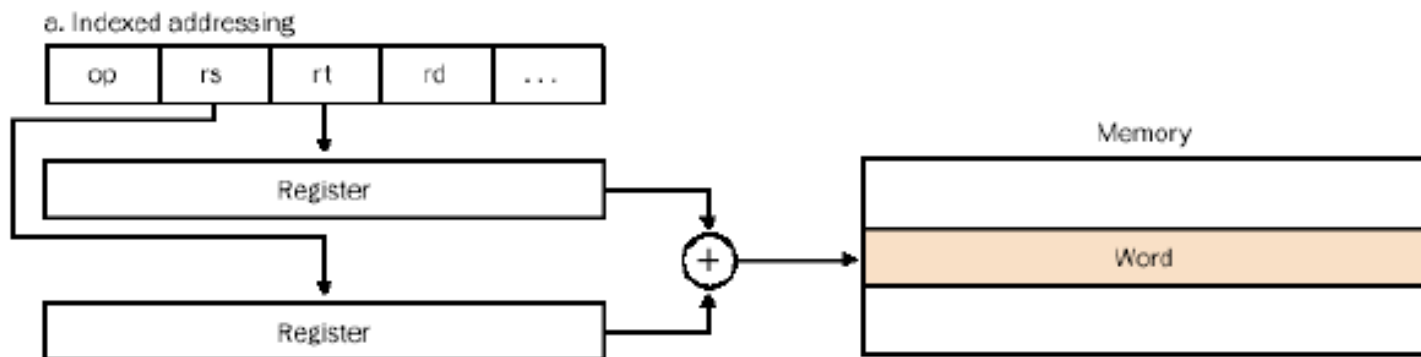
Exemplos “reais”

No MIPS:

```
add $t0, $a0, $s3 # a0 tem a base de um vetor, $s3 é  
                  # o índice  
lw  $t1, 0($t0)   # $t1 ← Memória[$a0 + $s3]
```

No PowerPC:

```
lw $t1, $a0 + $s3 # $t1 ← Memória[$a0 + $s3]
```



Considerações Finais

- As categorias de instruções do MIPS estão associadas às linguagens de alto nível:
 - Instruções aritméticas associadas aos comandos de atribuição.
 - Instruções de transferência de dados associadas a estruturas de dados como os vetores.
 - Desvios condicionais aos comando if e loops.
 - Desvios incondicionais associados a chamadas a procedimentos e retornos de procedimentos

Considerações Finais

- O ISA de uma máquina deve ter comprometimento entre a **quantidade de instruções** para a execução de um programa, **número de ciclos de clock** gasto por cada uma e a **frequência do clock**
- O balanceamento deve seguir quatro princípios básicos:
 1. *A simplicidade é favorecida pela regularidade:*
 - instruções de mesmo tamanho, campos dos registradores na mesma posição em cada formato de instrução e operações aritméticas sempre com três operandos
 2. *Quanto menor mais rápido:*
 - somente 32 registradores
 3. *Um bom projeto demanda compromisso:*
 - permitir a representação de constantes e endereços maiores e a necessidade de manter as instruções de mesmo tamanho
 4. *Torne o caso comum mais rápido:*
 - endereçamento relativo ao PC para desvios condicionais e endereçamento imediato para constantes.

Referências

- ❑ Patterson, David A.; Hennesy, John; Organização e projeto de computadores: a interface hardware/software; 3ª ed.; Elsevier, 2005.
- ❑ Prof. Luiz Henrique Andrade Correia; Notas de aula.