Universidade de Brasília
Departamento de Ciência da Computação

Assembly MIPS Procedimentos

Marcus Vinicius Lamar

Resumindo:

Operandos MIPS		
Nome	**Exemplo**	**Comentários**
32 Registradores	$s0-$s7, $t0-$t9, $zero, $a0-$a3, $v0-$v1, $gp, $fp, $sp, $ra, $at	Locais rápidos para dados. No MIPS, os dados precisam estar em registradores para a realização de operações aritméticas. O registrador MIPS $zero sempre é igual a 0. O registrador $at é reservado para o montador.

Resumindo:

Assembly do MIPS				
Categoria	**Instrução**	**Exemplo**	**Significado**	**Comentários**
Aritmética	add	add $s1,$s2,$s3	$s1 = $s2 + $s3	Três operandos; dados nos registr.
	subtract	sub $s1,$s2,$s3	$s1 = $s2- $s3	Três operandos; dados nos registr.
	add immediate	addi s1,$s2,100	$s1= $s2 + 100	Usada para somar constantes
Transferência de dados	load word	lw $s1,100($s2)	$s1 = Memória[$s2+100]	Dados da memória para o registr.
	store word	sw $s1,100($s2)	Memória[$s2 + 100] = $s1	Dados do reg para a memória
	load byte	lb $s1,100($s2)	$s1 = Memória[$s2 + 100]	Byte da memória para registrador
	store byte	sb $s1,100($s2)	Memória[$s2+100] = $s1	Byte de um registrador para memória
	load upper immed.	lui $s1,100	$s1 = 100 * 2^{16}	Carrega kte 16 bits mais altos
Desvio condicional	branch on equal	beq $s1,$s2,25	if ($s1==$s2)PC=PC+4+100	Testa ==; desvio relativo ao PC
	branch not equal	bne $s1,$s2,25	if ($s1!=$s2)PC=PC+4+100	Testa !=; desvio relativo ao PC
	set on less than	slt $s1,$s2,$s3	if ($s2 < $s3) $s1 = 1; else $s1 = 0	Compara menor que; usado com beq, bne
	set less than immediate	slti$s1,$s2,100	if ($s2 < 100) $s1 = 1; else $s1 = 0	Compara menor que constante
Desvio incondicional	jump	j 2500	go to 10000	Desvia para endereço de destino
	jump register	jr $ra	go to $ra	Para switch e retorno de função
	jump and link	jal 2500	$ra = PC + 4. go to 10000	Para chamada de função

Instruções de suporte a procedimentos

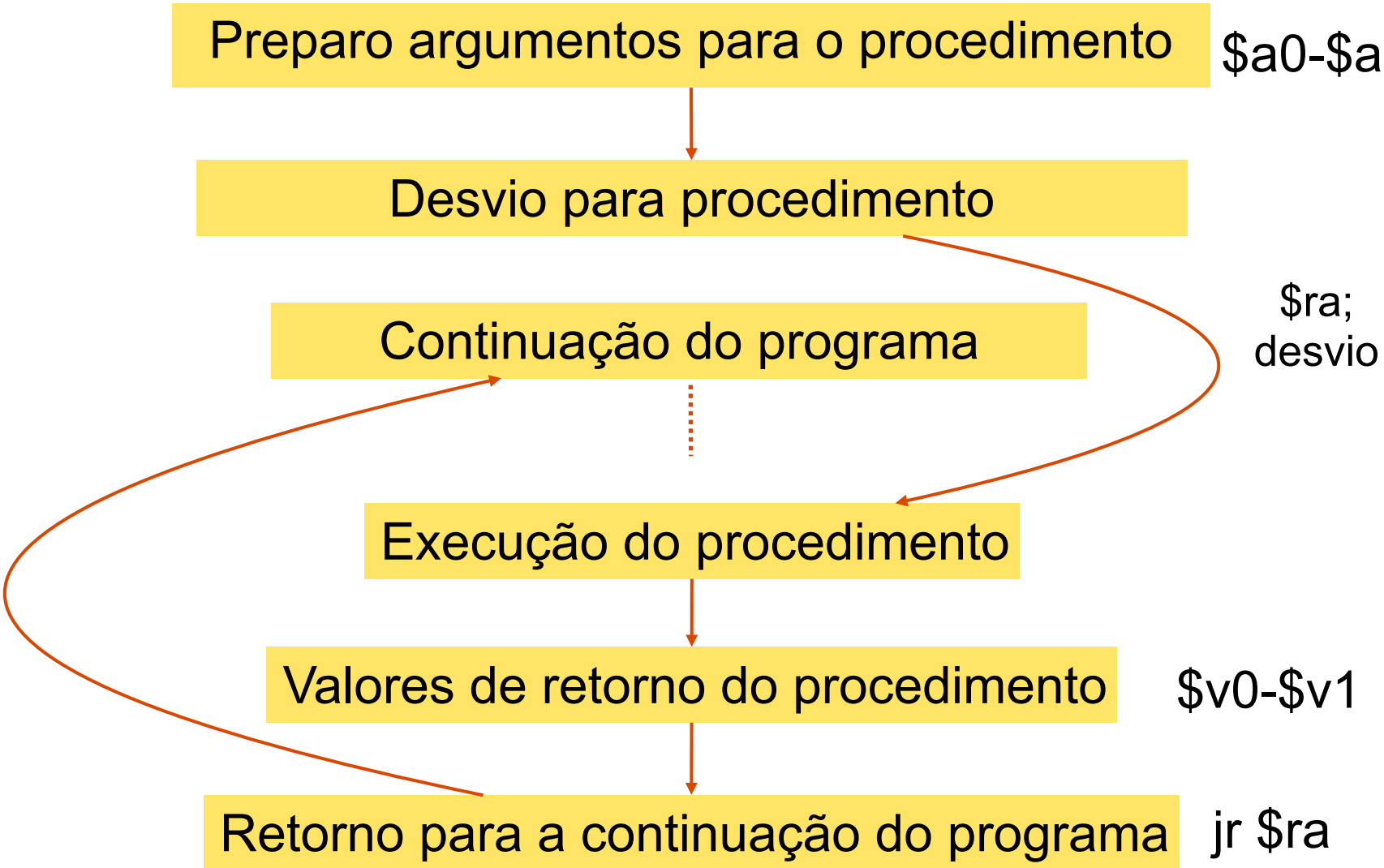
- Passos em um procedimento:
  1. Colocar os parâmetros em um lugar onde o procedimento possa acessá-los;
  2. Transferir o controle para o procedimento;
  3. Adquirir recursos de armazenamento necessários para o procedimento;
  4. Realizar a tarefa desejada;
  5. Colocar o valor de retorno em um lugar onde o programa que o chamou possa acessá-lo;
  6. Retornar o controle para o ponto de origem.

Instruções de suporte a procedimentos

- Qual o lugar mais rápido que pode armazenar dados em um computador?

- Registradores MIPS:
  - $a0 - $a3: parâmetros para os procedimentos;
  - $v0 - $v1: valores de retorno do procedimento;
  - $ra: registrador de endereço de retorno ao ponto de origem (*ra* = *return address*).

Instruções de suporte a procedimentos



jal (*jump and link*)

- *Link*, neste caso, quer dizer que é armazenada, no registrador *$ra*, o endereço da instrução que vem logo após a instrução *jal Label*:

Código “equivalente”

```
addi $ra, $PC, 4
j Label
```

- Por que existe a instrução jal?

Exemplo de procedimento

C

```
main()
{  …
   c=soma(a,b);...  /* a:=$s0; b:=$s1; c:=$s2 */
   …
}
   int soma(int x, int y)  /* x:=$a0;  y:=$a1 */
   {        return x+y; }
```

MIPS

```
   end
1000  add  $a0,$s0,$zero  # x = a
1004  add  $a1,$s1,$zero  # y = b
1008  jal  soma           # prepara $ra e j soma
1012  add $s2,$v0,$zero   # s2=a+b
…
2000  soma: add $v0,$a0,$a1
2004  jr   $ra            # volte p/ origem, 1012
```

Usando mais registradores

- Se precisar mais de 4 argumentos e 2 valores de retorno?.
- Se o procedimento necessitar utilizar registradores salvos $sx?

Usando mais registradores

- Como fazer isto?
- Processo conhecido por *register spilling*:
  - Uso de uma pilha;
  - Temos um apontador para o topo da pilha;
  - Este apontador é ajustado em uma palavra para cada registrador que é colocado na pilha (*push*), ou retirado da pilha (*pop*).
  - Em MIPS, o registrador 29 é utilizado somente para indicar o topo da pilha: $sp (*stack pointer*)

Usando a Pilha

- Por razões históricas, a pilha "cresce" do maior endereço para o menor endereço:
- Para colocar um valor na pilha (*push*), devemos decrementar *$sp* em uma palavra e mover o valor desejado para a posição de memória apontada por *$sp*;
- Para retirar um valor da pilha (*pop*), devemos ler este valor da posição de memória apontado por *$sp*, e então incrementar *$sp* em uma palavra.

Exemplo: exemplo_folha

- Suponha que tenhamos o seguinte código:

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

Vamos gerar o código correspondente em assembly MIPS.

Exemplo : exemplo_folha

- Definição: Os argumentos *g, h ,i e j* correspondem aos registradores *$a0, $a1, $a2* e *$a3*, e *f* corresponde a *$s0*.
- Definir o rótulo do procedimento:

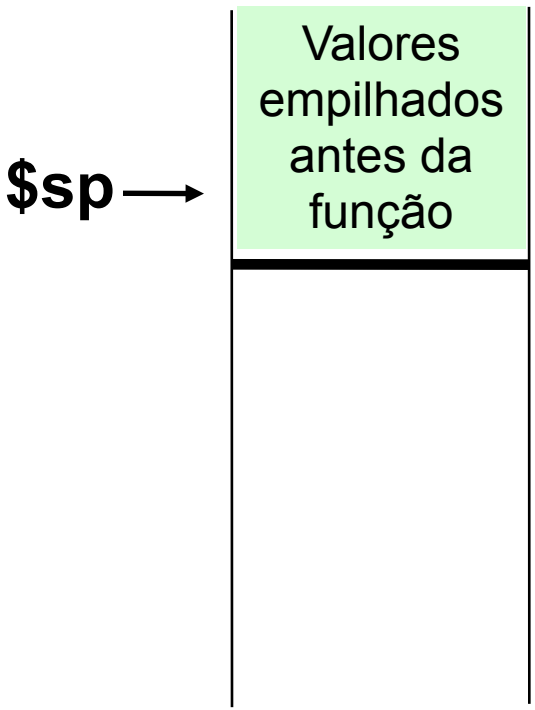
```
exemplo_folha:
```

- Devemos então armazenar na pilha os registradores que serão utilizados pelo procedimento:

```
addi $sp, $sp, -12      # cria espaço para 3 itens na pilha
sw $t1, 8($sp)          # empilha $t1
sw $t0, 4($sp)          # empilha $t0
sw $s0, 0($sp)          # empilha $s0
```

Exemplo : exemplo_folha

Como ficou a pilha?



Pilha antes da função

Pilha durante execução da função

Exemplo : exemplo_folha

- Corpo do procedimento:

```
add $t0, $a0, $a1          # $t0 = g + h
add $t1, $a2, $a3          # $t1 = i + j
sub $s0, $t0, $t1          # f = $s0 = (g+h) – (i+j)
```

- Resultado armazenado no registrador $v0:

```
add $v0, $s0, $zero        # retorna f em $v0
```

Exemplo : exemplo_folha

- Antes de sair do procedimento, restaurar os valores dos registradores salvos na pilha:

```
lw $s0, 0($sp )        # desempilha $s0
lw $t0, 4($sp)         # desempilha $t0
lw $t1, 8 ($sp)        # desempilha $t1
addi $sp, $sp, 12      # remove 3 itens da pilha
```

- Voltar o fluxo do programa para a instrução seguinte ao ponto em que a função exemplo_folha foi chamada:

```
jr $ra          # retorna para a subrotina que chamou
```

Versão Didática

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

```
exemplo_folha:
    addi $sp, $sp, -12    # cria espaço para 3 itens na pilha
    sw $t1, 8($sp)        # empilha $t1
    sw $t0, 4($sp)        # empilha $t2
    sw $s0, 0($sp)        # empilha $s0
    add $t0, $a0, $a1     # $t0 = g + h
    add $t1, $a2, $a3     # $t1 = i + j
    sub $s0, $t0, $t1     # f = $s0 = (g+h) – (i+j)
    add $v0, $s0, $zero # retorna f em $v0
    lw $s0, 0($sp)        # desempilha $s0
    lw $t0, 4($sp)        # desempilha $t0
    lw $t1, 8 ($sp)       # desempilha $t1
    addi $sp, $sp, 12     # remove 3 itens da pilha
    jr $ra                # retorna para a subrotina que chamou
```

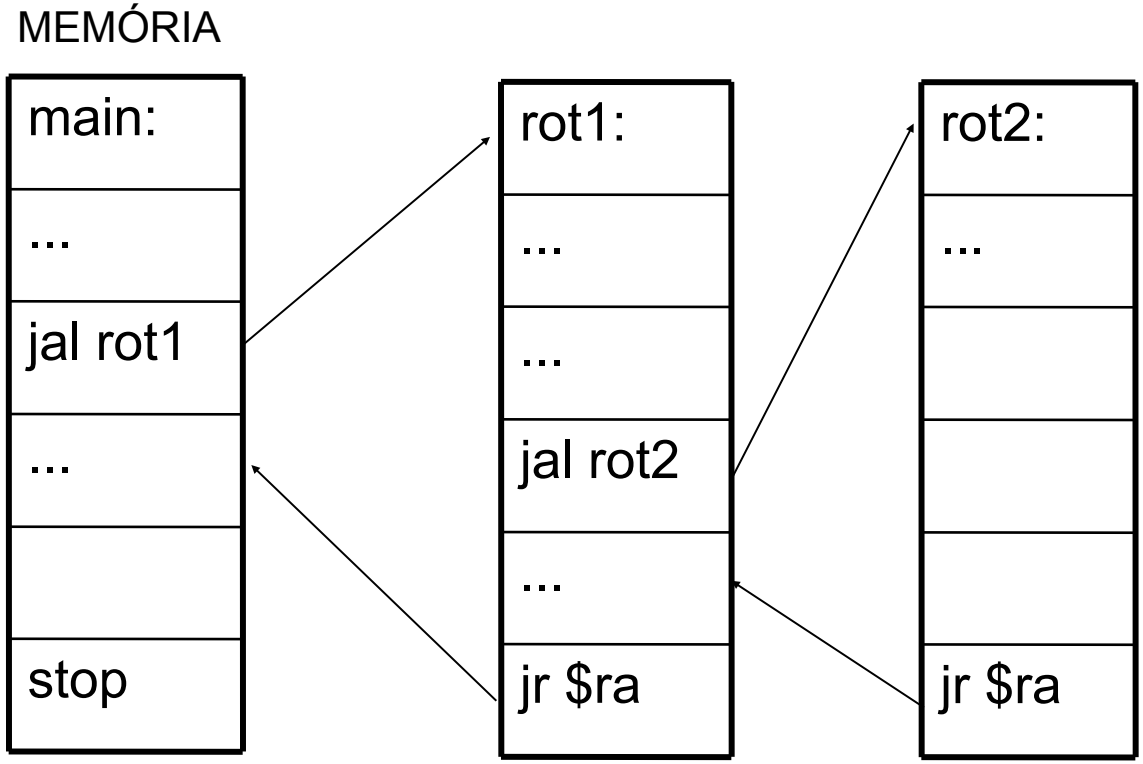
Versão otimizada

- Salvar o que realmente necessitar ser salvo
- Por convenção, os registradores t_i não precisam ser preservados.
- Utilizar registradores s_i onde realmente forem necessários.
- Ponderar uso de registradores com análise de desempenho.

```
exemplo_folha:
    add $v0, $a0, $a1    # $v0 = g + h
    sub $v0, $v0,$a2     # $v0 = g+h-i
    sub $v0, $v0, $a3    # f = $v0 = g+h-i-j
    jr $ra               # retorna para a subrotina que chamou
```

Procedimentos aninhados

- Suponha o seguinte procedimento aninhado:



- Problema: conflito com registradores $a e $ra!
- Como resolver?

Procedimentos aninhados: convenção sobre registradores

- Uma solução é empilhar todos os registradores que precisam ser preservados.
- Estabelecer uma convenção entre subrotinas chamada e chamadora sobre a preservação dos registradores (uso eficiente da pilha).
- Definições
  - Chamadora: função que faz a chamada, utilizando jal;
  - Chamada: função sendo chamada.

Por que utilizar convenções para chamadas de procedimentos?

- **Benefícios:**
  - programadores podem escrever funções que funcionam juntas;
  - Funções que chamam outras funções – como as recursivas – funcionam corretamente.

Exercício 1:

- Cálculo Matricial:

Implemente os seguintes procedimentos:

a) void Soma_Matriz(int destino[], int origem1[], int origem2[], int n);

b) void Mult_Matriz(int destino[], int origem1[], int origem2[], int n);

c) int Det_Matriz(int origem[], int n); /* n<4 */

d) void Show_Matriz(int origem[], int n);

Onde:

Ex.: int mat[]={1,2,3,4,5,6,7,8,9}; n=3;

$$mat = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

Exercício 2:

- Enésimo número primo:

Implemente um procedimento em Assembly MIPS que dado o argumento de entrada N, retorne o Nésimo número primo, e compile o programa principal abaixo.

```
void main()
{
        int n,np;
        printf("n=");
        scanf("%d",&n);
        np=primo(n);
        printf("o %d-ésimo primo é: %d\n",n,np);
}
```