Fundamentos de Arquitetura de Computadores

Tiago Alves

Faculdade UnB Gama
Universidade de Brasília

Revisão...

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.

Revisão...

Assembly do MIPS				
Categoria	Instrução	Exemplo	Significado	Comentários
Aritmética	add	add $s1,$s2,$s3	$s1 = $s2 + $s3	Três operandos; dados nos registradores
	subtract	sub $s1,$s2,$s3	$s1 = $s2- $s3	Três operandos; dados nos registradores
	add immediate	addi s1,$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 registrador
	store word	sw $s1,100($s2)	Memória[$s2 + 100] = $s1	Dados do registrador 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 constante nos 16 bits mais altos
Desvio condicional	branch on equal	beq $s1,$s2,25	if ($s1 == $s2) go to PC + 4 + 100	Testa igualdade; desvio relativo ao PC
	branch on not equal	bne $s1,$s2,25	if ($s1 != $s2) go to PC + 4 + 100	Testa desigualdade; 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 procedimento
	jump and link	jal 2500	$ra = PC + 4. go to 10000	Para chamada de procedimento

Instruções de suporte a procedimentos

Também conhecido como funções (com retorno ou tipo void)

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.

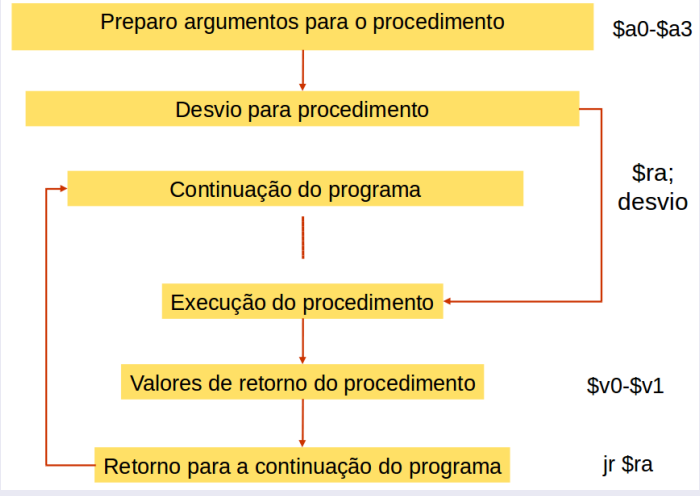
Instruções de suporte a procedimentos

Qual o lugar mais rápido em que se pode armazenar/manipular dados em um sistema eletrônico?

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



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`?

Instruções de suporte a procedimentos

C

```
main()
{   …
    c=soma(a,b);...  /* a:=$s0; b:=$s1; c:=$v0 */
    …
}

int soma(int x, int y)  /* x:=$a0;  y:=$a1 */
{ return x+y; }
```

MIPS

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

10

Usando mais registradores

Q: Se precisar mais de 4 argumentos e 2 valores de retorno?.

Q: Se o procedimento necessitar utilizar registradores salvos $sx?

A: 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).

A Pillha

- 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 de procedimentos folha (que não chamam outros procedimentos).

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.

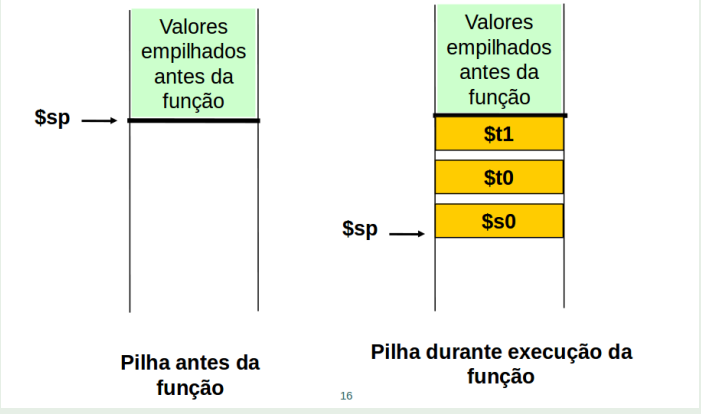
Exemplo de procedimentos folha (que não chamam outros procedimentos).

- 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 de procedimentos folha (que não chamam outros procedimentos).

Comportamento da pilha:



Exemplo de procedimentos folha (que não chamam outros procedimentos).

- 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 de procedimentos folha (que não chamam outros procedimentos).

- 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
```

Exemplo de procedimentos folha (que não chamam outros procedimentos).

Versão didática:

```
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
```

Exemplo de procedimentos folha (que não chamam outros procedimentos).

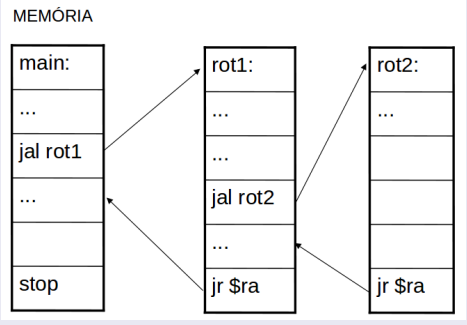
Versão não-didática:

- Salvar o que realmente necessitar ser salvo
- **Registradores $tx não precisam ser preservados**.
- Utilizar registradores $sx **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 $ax 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(Caller): função que faz a chamada, utilizando `jal`;
  - Chamada(Callee): função sendo chamada/receptora.

Procedimentos aninhados

Benefícios:

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

Exemplo: soma_recursiva

- Suponha que tenhamos o seguinte código, que calcula a soma n + (n-1) + ... + 2 + 1 de forma recursiva:

```
int soma_recursiva (int n)
{
  if (n < 1)
    return 0;
  else
    return n + soma_recursiva(n-1)
}
```

- Vamos gerar o código correspondente em assembly MIPS.

Exemplo: soma_recursiva

- O parâmetro n corresponde ao registrador $a0.
- Devemos inicialmente colocar um rótulo para a função, e salvar o endereço de retorno $ra e o parâmetro $a0:

```
Soma_recursiva:
 addi $sp, $sp, -8 # prepara a pilha para receber 2 itens
 sw $ra, 4($sp) # empilha $ra  (End. Retorno)
 sw $a0, 0($sp) # empilha $a0 (n)
```

- Na primeira vez que soma_recursiva é chamada, o valor de $ra que é armazenado corresponde ao endereço que está na rotina chamadora.

Exemplo: soma_recursiva

- Vamos agora compilar o corpo da função. Inicialmente, testamos se n < 1:

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

- Se n < 1, a função deve retornar o valor 0. Não podemos nos esquecer de restaurar a pilha.

```
add $v0, $zero, $zero    # valor de retorno é 0
addi $sp, $sp, 8         # remove 2 itens da pilha
jr $ra                   # retorne para depois de jal
```

- Por que não carregamos os valores de $a0 e $ra antes de ajustar $sp?

Exemplo: soma_recursiva

- Se n >=1, decrementamos n e chamamos novamente a função soma_recursiva com o novo valor de n.

```
L1: addi $a0, $a0, -1   # argumento passa a ser (n-1)
    jal soma_recursiva  # calcula a soma para (n-1)
```

- Quando a soma para (n-1) é calculada, o programa volta a executar na próxima instrução. Restauramos o endereço de retorno e o argumento anteriores, e incrementamos o apontador de topo de pilha:

```
lw $a0, 0($sp)   # restaura o valor de n
lw $ra, 4($sp)   # restaura o endereço de retorno
addi $sp, $sp, 8 # retira 2 itens da pilha.
```

Exemplo: soma_recursiva

- Agora o registrador $v0 recebe a soma do argumento antigo $a0 com o valor atual em $v0 (soma_recursiva para n-1):

```
add $v0, $a0, $v0 # retorne n + soma_recursiva(n-1)
```

- Por último, voltamos para a instrução seguinte à que chamou o procedimento:

```
jr $ra # retorne para a chamadora
```

Exemplo: soma_recursiva

```
Soma_recursiva:
  addi $sp, $sp, -8 # prepara a pilha para receber 2 itens
  sw $ra, 4($sp) # empilha $ra  (End. Retorno)
  sw $a0, 0($sp) # empilha $a0 (n)
  slti $t0, $a0, 1 # testa se n < 1
  beq $t0, $zero, L1              # se n>=1, vá para L1
  add $v0, $zero, $zero # valor de retorno é 0
  addi $sp, $sp, 8 # remove 2 itens da pilha
  jr $ra                # retorne para depois de jal

L1:
  addi $a0, $a0, -1 # argumento passa a ser (n-1)
  jal Soma_recursiva # calcula a soma para (n-1)
  lw $a0, 0($sp) # restaura o valor de n
  lw $ra, 4($sp) # restaura o endereço de retorno
  addi $sp, $sp, 8 # retira 2 itens da pilha.
  add $v0, $a0, $v0 # retorne n + soma_recursiva(n-1)
  jr $ra # retorne para a chamadora
```

O que deve ser preservado?

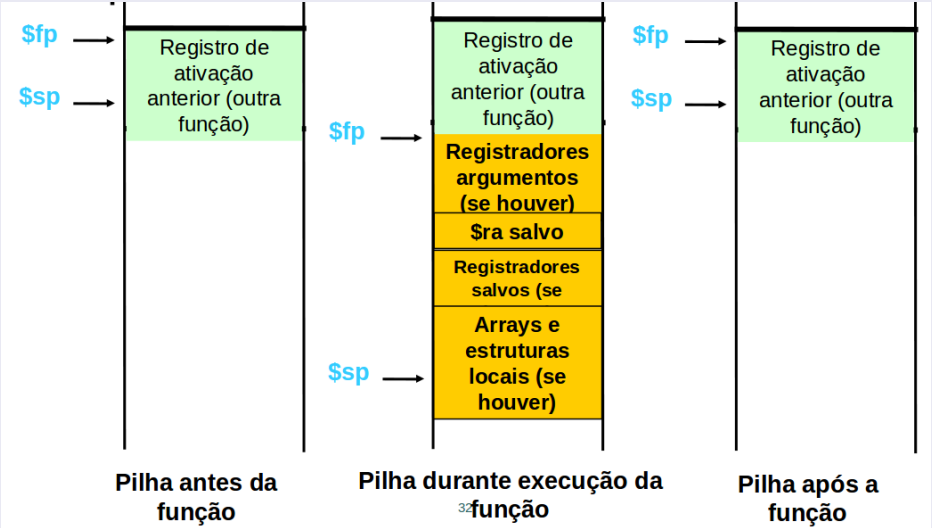
Preservado	Não Preservado
Registradores $s0-$s7	Registradores $t0-$t9
Stack Pointer $sp	Registradores $a0-$a3
Pilha acima do $sp	Pilha abaixo do $sp
Registrador de retorno $ra	Registradores $v0-$v1
Frame Pointer ($fp) Global Pointer ($gp) se utilizados	

Alocando espaço para novos dados (locais) na pilha

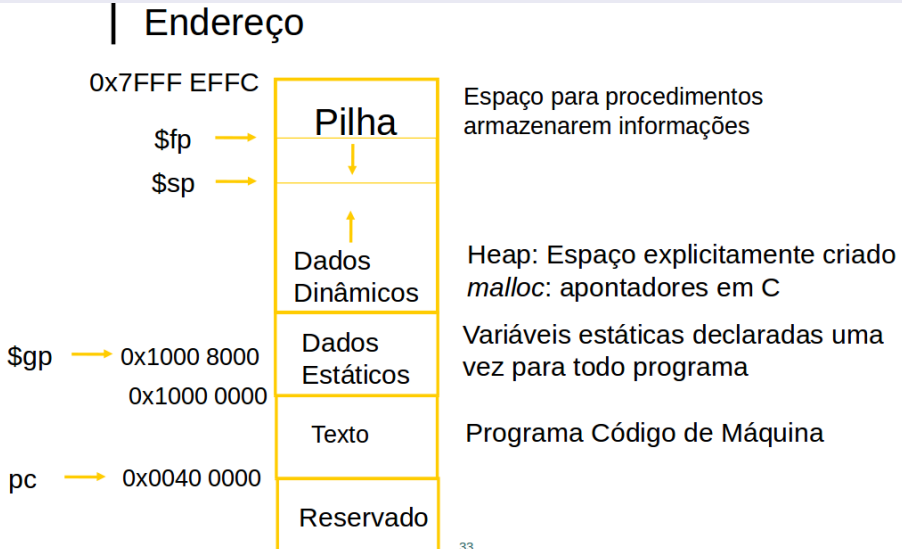
Frame de procedimento (Registro de ativação)

- Armazenar variáveis locais a um procedimento
- Facilita o acesso a essas variáveis locais ter um apontador estável $fp

Frame de procedimento



Alocação de memória (SPIM e MARS)



Política de Convenção de Uso dos Registradores

Nome	Número do registrador	Uso
$zero	0	O valor constante 0
$v0-$v1	02/03	Valores para resultados e avaliação de expressões
$a0-$a3	04/07	Argumentos
$t0-$t7	08/15	Temporários
$s0-$s7	16-23	Valores salvos
$t8-$t9	24-25	Mais temporários
$gp	28	Ponteiro global
$sp	29	Ponteiro de pilha
$fp	30	Pointeiro de quadro
$ra	31	Endereço de retorno

Registrador 1 ($at) reservado para o assembler, 26-27 para o sistema operacional

Exercício 1

Implemente uma rotina que calcule o enésimo valor da Série de Fibonacci

$1, 1, 2, 3, 5, 8, 13, 21, 34, 55, ...$

```
Fibonacci:      # $a0=n  $v0=Fibonacci(n)
```

.... onde Fibonacci(0)=1, Fibonacci(1)=1

Uma solução recursiva e uma não-recursiva.

Qual possui melhor desempenho?

Exercício 2

Cálculo matricial:

Implemente os seguintes procedimentos:

1. `void Soma_Matriz( int destino[], int origem1[], int origem2[], int n);`
2. `void Mult_Matriz(int destino[], int origem1[], int origem2[], int n);`
3. `int Det_Matriz(int origem[], int n);` `/* n<4 */`
4. `void Show_Matriz(int origem[], int n);`

Onde `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 3

N-ésimo número primo:

Implemente os seguintes procedimentos:

- 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-esimo primo e: %d\n",n,np);
}
```