Instruções MIPS

Funções e Procedimentos

Como funcionam os procedimento no MIPS?

Para manipular os procedimentos fazemos uso dos seguintes registradores:

- **$a0 até $a3:** são os registradores de argumentos utilizados para a passagem de parâmetros;
- **$v0 e $v1:** são os registradores de valor utilizados para o retorno do procedimento;
- **$ra:** é o registrador de endereço do retorno do procedimento, utilizado na volta ao ponto de origem da chamada do procedimento.

Também são necessários os **conceitos e instruções:**

JAL: é uma instrução de salto (jump) utilizada unicamente para os procedimentos (jump and link). Essa instrução desvia para um endereço e, ao mesmo tempo, salva o endereço da instrução seguinte no registrador de endereço de retorno. Sintaxe da instrução:

JAL endereco_procedimento

Exemplo:

jal fatorial

onde fatorial é o LABEL de um bloco de procedimentos e, relembrando, um LABEL é um nome dado a um endereço de memória.

JR: é uma instrução de salto, uma instrução de desvio incondicional para o endereço especificado em um registrador, ela volta ao endereço de retorno correto que é armazenado em $ra. Sintaxe da instrução:

jr $ra

Caller: é o programa que chama o procedimento, fornecendo os valores dos parâmetros. Faz uso dos registradores de **$a0 a $a5** para armazenar os parâmetros e também de **jal** para pular para o procedimento. Importante lembrar que o contador de programa é usado para realizar os cálculos dos endereços corretamente.

Callee: é um procedimento que executa uma série de instruções armazenadas com base nos parâmetros fornecidos pelo **Caller** e depois retorna o controle para o Caller novamente.

Spilled Registers: é o processo de colocar variáveis menos utilizadas na memória (ou variáveis que serão necessárias mais adiante).

Pilha: da mesma forma que uma pilha de chamadas e retornos de procedimentos é criada para linguagens como C e Java, aqui no MIPS também faremos uso deste conceito para gerenciar os Spilled Registers. Isto é necessário pois os registradores usados pelo Caller devem ser restaurados aos seus valores anteriores a chamada do procedimento.

Stack Pointer: é um valor que indica o endereço alocado mais recentemente em uma pilha, mostrando onde devem ser localizados os valores antigos dos registradores e onde os Spilled Registers devem ser armazenados. O registrador 29 é usado para o $sp.

Push: coloca palavras para cada registrador salvo ou restaurando na pilha. Valores são colocados na pilha pela subtração do valor do stack pointer.

Pop: remove palavras da pilha. Valores são retirados da pilha pela soma do valor do stack pointer.

Compilando um procedimento

```
1  int exemplo ( int g, int h, int i, int j) {
2      int f;
3      f = ( g + h ) - ( i + j );
4      return f;
5  }
```

Agora precisamos salvar os registradores temporários usados pelo corpo da função, isto é, **$t0 = (g + h)** e **$t1 = (i + j)**, que ficará da seguinte forma:

```
1  addi $sp, $sp, -12
2  sw $t1, 8 ($sp)
3  sw $t0, 4 ($sp)
4  sw $s0, 0 ($sp)
```

Vamos identificar os registradores:

- $a0 = g
- $a1 = h
- $a2 = i
- $a3 = j
- $s0 = f

O rótulo (label) no MIPS será “exemplo:” que é o nome da função em C

Compilando um procedimento

```
1  addi $sp, $sp, -12
2  sw $t1, 8 ($sp)
3  sw $t0, 4 ($sp)
4  sw $s0, 0 ($sp)
```

- Os valores antigos são empilhados de forma a criar espaço para três palavras de 12 bytes na pilha.
- A primeira linha ajusta a pilha criando um espaço para três itens, (quando vamos empilhar precisamos subtrair, pois a pilha cresce de endereços maiores para menores).
- A segunda linha do código MIPS salva o registrador $t1 para usar depois, o mesmo acontece com a terceira e quarta linha, elas salvam os registradores temporários para uso posterior.
- Notem que usamos corretamente o alinhamento da memória, pulando de 4 em 4 no endereço.

Continuando, devemos fazer a compilação do corpo da função:

```
1   add $t0, $a0, $a1 # (g + h)
2   add $t1, $a2, $a3 # (i + j)
3   sub $s0, $t0, $t1 # (g + h ) - ( i + j)
```

ATENÇÂO!!!
Aqui em vez de usarmos registradores $s e $t para g, h, i e j, nós usamos os registradores de parâmetros $a pois são os parâmetros passados na função que desejamos usar no cálculo.

Agora última coisa que está faltando é o retorno da função e para isso fazemos:

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

Continuando, devemos fazer a compilação do corpo da função:

```
1   add $t0, $a0, $a1 # (g + h)
2   add $t1, $a2, $a3 # (i + j)
3   sub $s0, $t0, $t1 # (g + h ) - ( i + j)
```

ATENÇÂO!!!
Aqui em vez de usarmos registradores $s e $t para g, h, i e j, nós usamos os registradores de parâmetros $a pois são os parâmetros passados na função que desejamos usar no cálculo.

Agora última coisa que está faltando é o retorno da função e para isso fazemos:

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

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

Copiamos f para um registrador de valor de retorno, neste caso $v0, e antes de retornar para o ponto de origem, os valores antigos devem ser restaurados usando para isto o desempilhamento:

```
1 lw $s0, 0 ($sp)
2 lw $t0, 4 ($sp)
3 lw $t1, 8 ($sp)
4 addi $sp, $sp, 12
```

```
1 lw $s0, 0 ($sp)
2 lw $t0, 4 ($sp)
3 lw $t1, 8 ($sp)
4 addi $sp, $sp, 12
```

As três primeiras linhas restauram os registradores para o caller e a última linha ajusta a pilha para excluir os 3 itens.

O procedimento deve terminar com:

```
1 jr $ra
```

Importante ressaltar que $s0 a $s7 são preservados em uma chamada de procedimento enquanto que registradores de $t0 a $t9 não são.

O código completo para o MARS fica da seguinte forma:

```
.text
li $a0, 15 # g = $a0 = 15
li $a1, 20 # h = $a1 = 20
li $a2, 10 # i = $a2 = 10
li $a3, 5 # j = $a3 = 5

exemplo:
addi $sp, $sp, -12
sw $t1, 8 ($sp)
sw $t0, 4 ($sp)
sw $s0, 0 ($sp)

add $t0, $a0, $a1 # (g + h) = 15 + 20
= 35 (hexa = 23)
add $t1, $a2, $a3 # (i + j) = 10 + 5
= 15 (hexa = F)
sub $s0, $t0, $t1 # (g + h ) - ( i +
j) = 35 - 15 = 5 ( hexa = 5)
add $v0, $s0, $zero # f ($v0 = $s0 +
0 )

lw $s0, 0 ($sp)
lw $t0, 4 ($sp)
lw $t1, 8 ($sp)
addi $sp, $sp, 12

jr $ra
```

Exercício para entrega (Moodle)

Suponha que você já tenha escrito uma função MIPS com a seguinte assinatura:

int sum (int A [], int primeiro, int último).

Esta função calcula a soma dos elementos de A começando com o primeiro elemento e terminando com o último elemento.

Escreva um programa em linguagem assembly MIPS que chama essa função e a utiliza para calcular a média de todos os valores em A. Você pode assumir que o tamanho da matriz A é N, o endereço base de A está em $a0.

Register Number	Conventional Name	Usage
$0	$zero	Hard-wired to 0
$1	$at	Reserved for pseudo-instructions
$2 - $3	$v0, $v1	Return values from functions
$4 - $7	$a0 - $a3	Arguments to functions - not preserved by subprograms
$8 - $15	$t0 - $t7	Temporary data, not preserved by subprograms
$16 - $23	$s0 - $s7	Saved registers, preserved by subprograms
$24 - $25	$t8 - $t9	More temporary registers, not preserved by subprograms
$26 - $27	$k0 - $k1	Reserved for kernel. Do not use.
$28	$gp	Global Area Pointer (base of global data segment)
$29	$sp	Stack Pointer
$30	$fp	Frame Pointer
$31	$ra	Return Address
$f0 - $f3	-	Floating point return values
$f4 - $f10	-	Temporary registers, not preserved by subprograms
$f12 - $f14	-	First two arguments to subprograms, not preserved by subprograms
$f16 - $f18	-	More temporary registers, not preserved by subprograms
$f20 - $f31	-	Saved registers, preserved by subprograms