Linguagem de montagem

5. Procedimentos

Prof. John L. Gardenghi
Adaptado dos slides do livro

Chamada a procedimentos

- Conceitos preliminares
  - *caller*: programa que chama o procedimento
  - *callee*: procedimento chamado
  - *program counter (PC)*: registrador especial que possui o endereço de memória da instrução que o processador está executando
  - *Procedimento folha*: um procedimento que não faz chamada a algum outro (*alusão do nome: chamada de procedimentos como uma árvore*).

Chamada a procedimentos

- Fluxo de chamada e execução de procedimentos
  1. Coloque os parâmetros nos registradores
  2. Desvie a execução para o procedimento
  3. Ajuste o armazenamento para o procedimento
  4. Execute o procedimento
  5. Salve o resultado no registrador de retorno
  6. Retorne ao procedimento chamador

Uso dos registradores

- $a0 – $a3: argumentos (reg's 4 – 7)
- $v0, $v1: valores de retorno (reg's 2 and 3)
- $t0 – $t9: temporários
  - Podem ser sobrescritos pelo *callee*
- $s0 – $s7: salvos
  - Devem ser salvos e restaurados pelo *callee*
- $gp: ponteiro global para dados estáticos (r. 28)
- $sp: ponteiro para a pilha (reg 29)
- $fp: ponteiro para o *frame* (reg 30)
- $ra: endereço de retorno (reg 31)

Instruções para chamadas

- **Chamada a procedimento**: *jump and link*

  `jal ProcedureLabel`

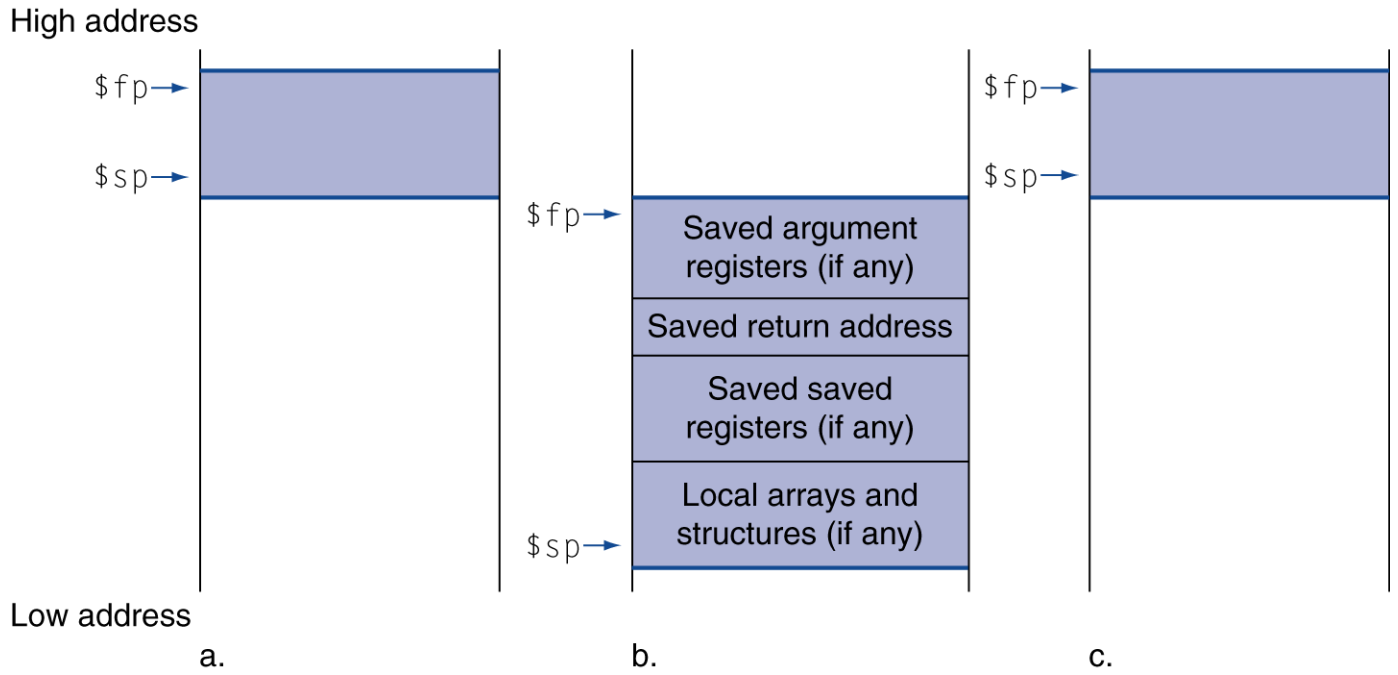
  - Endereço da próxima instrução salvo em $ra
  - Desvia para `ProcedureLabel`

- **Retorno do procedimento**: *jump register*

  `jr $ra`

  - Copia o conteúdo de $ra para o PC
  - Desvia o fluxo de volta ao *caller*

A pilha



- Dados locais do *callee*
  - e.g., variáveis automáticas do C
- Frame de procedimento (registro de ativação)
  - Espaço entre $fp e $sp
  - Usado por alguns compiladores para gerenciar o armazenamento de um procedimento

Usando a pilha

- Como usar a pilha num procedimento
  1. Alocar espaço na pilha
  2. Salvar os dados
  3. Restaurar os dados
  4. Desalocar espaço na pilha
- Como alocar espaço na pilha
  - Subtrair 4 do registrador $sp para cada palavra a ser armazenada
  - Ex: `addi $sp, $sp, -8` aloca espaço para duas palavras na pilha

Usando a pilha

- Como salvar dados na pilha
  - Salvar palavra por palavra em cada posição alocada
  - Deslocar `4 x posição` no `$sp`
- Exemplo
  - `addi $sp, $sp, -8 # aloca espaço para 2 pal.`
  - `sw $s0, 4($sp)  # salva $s0 na posição 2`
  - `sw $s1, 0($sp)  # salva $s1 na posição 1`

Exemplo de proc. folha

- Código C:

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

  - Argumentos g, …, j em $a0, …, $a3
  - f em $s0 (importante: salvar $s0 na pilha)
  - Resultado em $v0

Exemplo de proc. folha

- Código MIPS:

```
exemplo_folha:
   addi $sp, $sp, -4
   sw   $s0, 0($sp)
   add  $t0, $a0, $a1
   add  $t1, $a2, $a3
   sub  $s0, $t0, $t1
   add  $v0, $s0, $zero
   lw   $s0, 0($sp)
   addi $sp, $sp, 4
   jr   $ra
```

Aloca 1 posição na pilha

Salva $s0 na pilha

Operações do procedimento

Resultado

Restaura o valor de $s0

Desaloca o espaço na pilha

Retorna ao *caller*

Procedimentos não-folha

- Procedimentos que chamam outros
- Antes de chamar outro procedimento, é necessário salvar
  - O endereço de retorno ($ra)
  - Quaisquer argumentos e temporários necessários após a chamada
- Restaura dados da pilha depois da chamada

Exemplo de proc. não-folha

- Código C:

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

  - Argumento n em $a0
  - Resultado no $v0

Exemplo de proc. não-folha

- Código MIPS:

```
fat:
    addi $sp, $sp, -8       # adjust stack for 2 items
    sw   $ra, 4($sp)        # save return address
    sw   $a0, 0($sp)        # save argument
    slti $t0, $a0, 1        # test for n < 1
    beq  $t0, $zero, L1
    addi $v0, $zero, 1      # if so, result is 1
    addi $sp, $sp, 8        #   pop 2 items from stack
    jr   $ra                #   and return
L1: addi $a0, $a0, -1       # else decrement n
    jal  fact               # recursive call
    lw   $a0, 0($sp)        # restore original n
    lw   $ra, 4($sp)        #   and return address
    addi $sp, $sp, 8        # pop 2 items from stack
    mul  $v0, $a0, $v0      # multiply to get result
    jr   $ra                # and return
```

Padrão da memória

- Text: código de programas
- Static data: variáveis globais
  - e.g., variáveis estáticas em C, arrays de tamanho fixo e strings
  - $gp é inicializado “no meio” para permitir offsets positivos e negativos
- Dados dinâmicos: heap
  - E.g., malloc em C, new no Java
- Stack: pilha