Registros de Ativação

Sandro Rigo
sandro@ic.unicamp.br

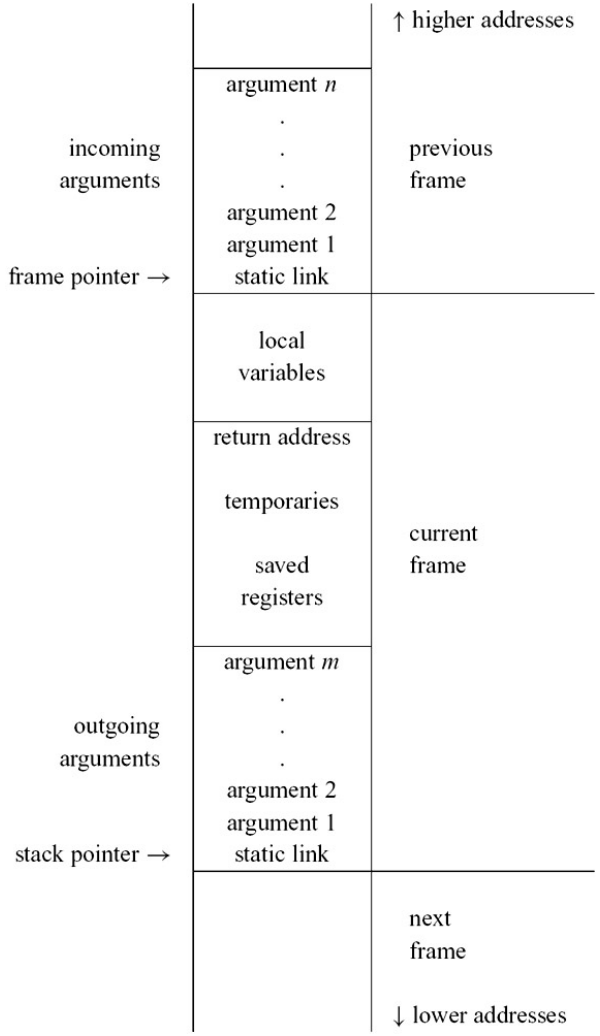
Introdução

- Funções têm variáveis locais
  - Devem ser criadas na chamada da função
  - Sobrevivem até que a função retorne (C, Pascal, Java)
- Recursão
  - Cada instância da função tem seus próprios parâmetros e locais
- Chamadas de funções
  - Se comportam de maneira LIFO
  - Podemos usar uma pilha

Stack Frames

- As operações push e pop não podem ser feitas individualmente para cada variável
  - Manipula-se conjuntos de variáveis
  - Precisamos ter acesso a todas elas
- Stack Pointer (SP)
  - Todas as posições além do SP são lixo
  - Todas as anteriores estão alocadas
- Activation Record ou Stack Frame
  - Área na pilha reservada para os dados de uma função
    - parâmetros, locais, endereço de retorno, etc

Stack Frames



- A pilha normalmente cresce para baixo
- O formato do frame depende
  - Linguagem sendo compilada
  - Características do processador alvo
  - Normalmente o fabricante do processador determina um layout padrão
    - Funções escritas numa linguagem podem chamar funções de outra linguagem

Frame Pointer

- Suponha que g() chama f(a1, a2, …,an)
  - g() é conhecida como caller (chamador)
  - f() é conhecida como callee (chamado)
- Na chamada de f()
  - SP aponta para o primeiro argumento sendo passado a f()
  - f() aloca seu frame subtraindo o tamanho de SP
- O antigo SP se torna o atual FP
- Em algumas arquiteturas o FP é um registrador
  - Seu valor antigo é salvo no frame e restaurado no retorno

Frame Pointer

- FP é útil quando o tamanho dos frames pode variar ou não são contíguos na pilha
- Com frames de tamanho fixo:
  - O FP sempre diferirá de SP por um tamanho conhecido
  - Não é necessário gastar um registrador para isso
- Por que um FP?
  - O tamanho do frame só pode ser calculado muito adiante no processo de compilação
    - É necessário saber o número de temporários e registradores a serem salvos

Frame Pointer

- Por que um FP?
  - Porém é útil saber os offsets dos parâmetros e locais
  - São alocados primeiro, próximos ao FP
    - Offset conhecido mais cedo

Registradores

- Registradores são unidades de armazenamento internas do processador
  - Tempo de acesso é milhares de vezes mais rápido que a memória
- O bom uso dos registradores é essencial para um bom desempenho do programa
- O número de registradores é limitado. Ex:32
- É comum instruções aritméticas poderem acessar valores somente em registradores
  - RISC

Registradores

- Muitas funções podem precisar dos registradores ao mesmo tempo
- Suponha que f() chama g() e ambas usam o registrador r
  - Caller-save: É responsabilidade de f salvar e restaurar o registrador r
  - Callee-save: É responsabilidade de g salvar e restaurar o registrador r
- Normalmente não existe uma diferença física entre os registradores
  - Diferenciação entre caller e callee saves é convenção da arquitetura

Registradores

- Uma boa escolha entre caller e callee save registers pode economizar acessos à memória
- Exemplo: No MIPS
  - $t0-$t9: caller-saves
  - $s0-$s7: callee-saves

Parâmetros

- Estudos mostram que quase todas têm no máximo 4 argumentos
- Quase nenhuma tem mais que 6
- Antigamente, passagem era sempre feita na pilha
  - Tráfego de memória desnecessário
- Hoje em dia:
  - Primeiros k parâmetros passados em registradores, o resto na pilha
    - K = 4 ou 6 são valores típicos

Endereço de Retorno

- Antigamente era colocado na pilha no momento da chamada da função
- Atualmente é mais comum ser mantido em registrador
  - No MIPS: ra ou $31
- Procedimentos que não são folhas devem salvá-lo na pilha

Variáveis na Pilha

- Razões para alocar uma variável na pilha:
  - Passagem por referência (& em C)
  - Variáveis acessadas por funções aninhadas
  - Valor muito grande para caber em um registrador
    - Alguns compiladores podem dividí-lo
  - Registrador ocupado pela variável se torna necessário para outro propósito
  - Spill: há variáveis locais e temporários demais para caber todos em registradores, alguns são forçadamente colocados na pilha

Static Links

- Funções Aninhadas:
  - Funções internas podem acessar varíáveis das externas
- Static Link: um ponteiro para o frame da função que estaticamente engloba a chamada
- Exemplo: programa 6.3 do livro do Appel

Prog 6.3: Funções Aninhadas

```
type tree = {key: string, left: tree, right: tree}
function prettyprint(tree: tree) : string =
  let
     var output := ""
     function write(s: string) =
        output := concat(output,s)
     function show(n:int, t: tree) =
        let function indent(s: string) =
              (for i := 1 to n do write(" ");
                   output := concat(output,s); write("\n"))
        in if t=nil then indent(".")
           else (indent(t.key); show(n+1,t.left); show(n+1,t.right))
     end
   in show(0,tree); output
 end
```