
Registros de Ativação

Guido Araujo
guido@ic.unicamp.br

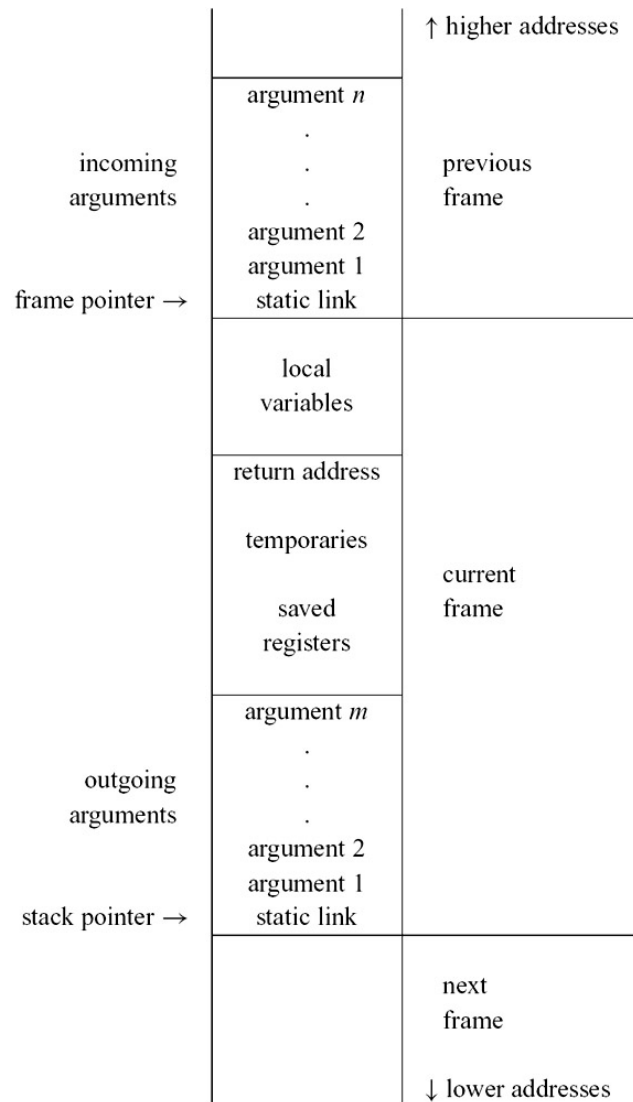
Introdução

- 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 $f()$ chama $g(a_1, a_2, \dots, a_n)$
 - $f()$ é conhecida como caller (chamador)
 - $g()$ é conhecida como callee (chamado)
- Na chamada de $g()$
 - SP aponta para o primeiro argumento sendo passado a $g()$
 - $g()$ 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

Parâmetros

- Estudos mostram que quase todas chamadas 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 é um valor típico

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
- Como em qual alocar?

Callee-save (\$s0-\$s7)

```
f ( ....) {  
$s1 x = y + 1  
$s1 = x  
g (a1, a2, ...)  
x = $s1  
y = 2 * x $s1  
}
```

```
g (a1,a2,...) {
```

\$s1 not used here

```
}
```

```
g (a1,a2,...) {  
mem = $s1
```

```
$s1 = 2 * $2
```

```
$s1 = mem
```

```
}
```

Caller-save (\$t0-\$t9)

\$t1 → f (....) {
 x = y + 1
 ~~\$t1 = x~~
 g (a1, a2, ...)
}

x not used here

g (a1,a2,...) {

\$t1 = 2 * \$t2

}

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 dividi-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 variáveis das externas
- Static Link: um ponteiro para o frame da função que estaticamente engloba a chamada
- Exemplo: programa 6.3 do livro