Funções não folha e recursão

Yuri Kaszubowski Lopes

UDESC

Anotações

Funções folha e não-folha

Funções folha

- A função apresentada na aula passada é uma função folha
- Uma função que realiza sua tarefa e retorna sem chamar outra função

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

Funções **não** folha

- Uma função que chama outra internamente para resolver o problema é uma função não-folha
- A função pode chamar outra função, ou um clone de si mesma (recursão)
  - Os problemas são os mesmos em ambos os casos

Anotações

Revisão: Funções folha

- Em funções folha podemos evitar (ou diminuir) o uso da pilha
  - Não é necessário escrever em `$ra`
  - Podemos usar registradores `$t_`

```
1  leaf_example:
2      addi $sp, $sp, -8
3      sw $s0, 0($sp)
4      sw $s1, 4($sp)
5      add $s0, $a0, $a1
6      add $s1, $a2, $a3
7      sub $v0, $s0, $s1
8      lw $s0, 0($sp)
9      lw $s1, 4($sp)
10     addi $sp, $sp, 8
11     jr $ra  # saltando para o endereço armazenado em $ra
```

Anotações

Revisão: Funções folha

- Em funções folha podemos evitar (ou diminuir) o uso da pilha
  - Não é necessário escrever em `$ra`
  - Podemos usar registradores `$t_`

```
leaf_example:
    add $t0, $a0, $a1
    add $t1, $a2, $a3
    sub $v0, $t0, $t1
    jr $ra  # saltando para o endereço armazenado em $ra
```


Anotações

Revisão: Salvar ou não salvar

Preservado	Não preservado
`$s0` — `$s7`	`$t0` — `$t9`
`$sp`	`$a0` — `$a3`
`$ra`	`$v0` — `$v1`
Pilha acima de `$sp`	Pilha abaixo de `$sp`

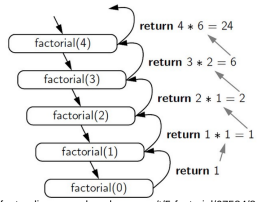
- Em funções não folha podemos não ter estas opções
- Uma função que chama outra função
  - Vamos escrever em `$ra` ao chamar a outra função com `jal`
  - Podemos ter situações que queremos preservar dados antes da chamada da outra função para usar após o retorno da outra função

Anotações

Funções não-folha

- Considere a função que calcula o fatorial recursivamente
  - Que problemas criamos agora a nível de linguagem de montagem que não existiam em uma função folha?
    - Os valores dos registradores podem se perder
    - O endereço de retorno em `$ra` vai se perder, e não saberemos voltar para a função original que chamou fatorial

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

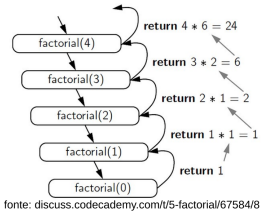


fonte: discuss.codecademy.com/t/5-factorial/67584/8

Anotações

Funções não-folha

- Cada chamada de fatorial deveria ter seus próprios registradores (e.g., `$s0`, `$s1`) e seu próprio retorno
  - Como podemos resolver?
  - Podemos mais uma vez empilhar os valores que precisam ser salvos



fonte: discuss.codecademy.com/t/5-factorial/67584/8

Anotações

Criação do Fatorial Recursivo

```
1     .text
2     .globl main
3 main:
4     li $a0, 4
5     jal fatorial
6     move $a0, $v0
7     li $v0, 1
8     syscall  # imprimir
9 end:
10    li $v0, 10
11    syscall
12 fatorial:
13    # vamos escrever nossa função aqui
```


Anotações

Criação do Fatorial Recursivo

```
12 fatorial:
13        # if ($a0 < 1)
14    blt $a0, 1, fat_parada
15    # salvar contexto
16
17    # código principal com chamada recursiva
18
19    # restaurar
20
21    # retornar
22
23 fat_parada:
24    ori $v0, $zero, 1
25    jr $ra
```


Anotações

Criação do Fatorial Recursivo

```
12 fatorial:
13         # if ($a0 < 1)
14     blt $a0, 1, fat_parada
15     # salvar contexto
16     addi $sp, $sp, -8
17     sw $s0, 0($sp)
18     sw $ra, 4($sp)
19     # código principal com chamada recursiva
20     move $s0, $a0
21     addi $a0, $a0, -1
22     jal fatorial  # $v0 = fatorial($a0 -1)
23     mul $v0, $v0, $s0  # $v0 *= $s0
24     # restaurar
25     lw $s0, 0($sp)
26     lw $ra, 4($sp)
27     addi $sp, $sp, 8
28     # retornar
29     jr $ra
30 fat_parada:
31     ori $v0, $zero, 1
32     jr $ra
```


Anotações

Notas

- Muitos problemas possuem soluções mais simples quando utilizado o conceito de recursividade
  - Navegar em uma árvore binária/grafo
  - Técnicas como guloso, programação dinâmica, divisão e conquista
  - A recursão (ou mesmo chamada de procedimentos não folha) custa caro para a máquina
    - Por quê?
    - Cada chamada exige comunicação com a memória para empilhar os dados
    - Ocupa espaço na pilha
    - Invalida nossa memória cache (veremos isso adiante)
  - Sendo assim, uma solução iterativa é preferível a nível de linguagem de máquina
  - Compiladores modernos fazem o que podem para tentar eliminar recursões

Anotações

Exercícios

1. Execute o fatorial recursivo no MARS passo a passo, analise e entenda as mudanças ocorridas em cada um dos registradores e nos endereços de memória.
2. Questões do laboratório de programação (notas das atividades)
   - O Moodle testa corretude
   - Será avaliado manualmente se atende o requisito de resolver recursivamente (nota zero na questão se os requisitos não forem cumpridos)
   - Será avaliado manualmente a qualidade do código: estilo, eficiência, elegância, limpeza, ...

Anotações

Referências

- D. Patterson; J. Henessy. **Organização e Projeto de Computadores: Interface Hardware/Software**. 5a Edição. Elsevier Brasil, 2017.
- Andrew S. Tanenbaum. **Organização estruturada de computadores**. 5. ed. São Paulo: Pearson, 2007.
- Harris, D. and Harris, S. **Digital Design and Computer Architecture**. 2a ed. 2012.
- courses.missouristate.edu/KenVollmar/mars/

Anotações

Anotações

Anotações