Rekursives Programmieren in Assembler

TechGI 2 - SoSe 2014

Vorgehen beim Übersetzen rekursiver Funkionen in Assembler

Jede rekursive Funktion besteht aus einer gewissen Rechnung und mindestens einem Anker.

Tipp: zuerst die Anker programmieren und danach die tatsächliche Funktion.

f(n) = 2+f(n-1)
f(0) = 0 ← Anker
	1.
	2.
	3.
	4.
	5.
	6.
	7.
	8.
		`# $a0 = n`
		`f:  beq $a0, $0, Anker  # n=0 -> Anker`
		
		`.`
		`.`
		`.`
		
		`Anker: addi $v0, $0, 0  # f(0) = 0`
		`jr $ra`

Datensicherung und Parameteranpassung vor dem rekursiven Aufruf

- Welche Register müssen gesichert werden?
- Wie müssen die Parameter angepasst werden?

In diesem Beispiel muss nur $ra gesichert werden.

Wiederherstellung der gesicherten Daten nach dem rekursiven Aufruf

Reihenfolge beim Sichern muss eingehalten werden.
Speicherfreigabe nicht vergessen.

f(n) = 2+f(n-1)
f(0) = 0
	1.
	2.
	3.
	4.
	5.
	6.
	7.
	8.
	9.
	10.
	11.
	12.
	13.
	14.
	15.
	16.
	17.
	18.
	19.
	20.
		`# $a0 = n`
		`f:  beq $a0, $0, Anker  # n=0 -> Anker`
		
		`addi $sp, $sp, -4  # Platz für 1`
		`# Register auf dem`
		`# Stack`
		
		`sw $ra, 0($sp)  # $ra sichern`
		
		`addi $a0, $a0, -1  # Parameter anpassen`
		`jal f  # rekursiver Aufruf`
		
		`lw $ra, 0($sp)  # $ra wiederherstellen`
		
		`addi $sp, $sp, 4  # Speicher wieder`
		`# freigeben`
		
		`jr $ra  # Rücksprung zum HP`
		
		`Anker: addi $v0, $0, 0  # f(0) = 0`
		`jr $ra`

Beispiel

Fibonacci-Funktion

$ra sichern

Zwischenergebnis sichern

Parameter für 2. Aufruf sichern

fib(n) = fib (n-1)+ fib (n-2)
fib (1) = 1
fib (0) = 0

```
    # $a0 = n
1.  fib:  beq $a0, $0, Anker0   # n=0-> Anker(n=0)
2.
3.        addi $t0, $0, 1       # 1
4.        beq $a0, $t0, Anker1  # n=1-> Anker(n=1)
5.
6.        addi $sp, $sp, -8     # Platz für 2 Reg.
7.        sw $ra, 0($sp)        # $ra sichern
8.        addi $s0, $a0, 0      # n->$s-Register
9.        sw $s0, 4($sp)        # n sichern
10.
11.       addi $a0, $a0, -1     # Parameter anpassen
12.       jal fib               # fib(n-1)
13.
14.       lw $ra, 0($sp)        # $ra laden
15.       lw $s0, 4($sp)        # n laden
16.       addi $a0, $s0, 0      # n->$a-Register
17.       addi $sp, $sp, 8      # Speicher freigeb.
18. # $a0 = n
19. # $v0 = fib(n-1)
20.       addi $sp, $sp, -12    # Platz für 3 Reg.
21.       sw $ra, 0($sp)        # $ra sichern
22.       addi $s0, $v0, 0      # fib(n-1)
23.                             #  -> $s-Register
24.       sw $s0, 4($sp)        # fib(n-1) sichern
25.       addi $s1, $a0, 0      # n -> $s-Register
26.       sw $s1, 8($sp)        # fib(n-1) sichern
27.
28.       addi $a0, $a0, -2     # Parameter anpassen
29.       jal fib               # fib(n-2)
30.
31.       lw $ra, 0($sp)        # $ra laden
32.       lw $s0, 4($sp)        # fib(n-1) laden
33.       addi $t0, $s0, 0      # fib(n-1) -> temp
34.       lw $s1, 8($sp)        # n laden
35.       addi $a0, $s1, 0
36.       addi $sp, $sp, 12     # Speicher freigeb.
37.
38. # $v0 = fib(n-2)
39. # $t0 = fib(n-1)
40.       add $v0, $t0, $v0     # fib(n-1)+fib(n-2)
41.
42.       jr $ra
43.
44. Anker0: addi $v0, $0, 0    # f(0) = 0
45.        jr $ra
46. Anker1: addi $v0, $0, 1    # f(1) = 1
47.        jr $ra
```