

Technische Informatik I

Uebung 1

Andreas Hejj, Benjamin Knecht, Pascal Gohl

Uebungsablauf:

Stunde 1: Einsteigende Kurztips, Tipps zu Aufgabe 1 und selbststaendiges lösen der Aufgabe, Besprechung der Lösung.

Stunde 2: Tipps zu Aufgabe 2 und selbststaendiges lösen der Aufgabe, Besprechung der Lösung.

Lernziele

Die Studentinnen und Studenten:

- kennen den **Aufbau** eines einfachen **Assembler-Programmes**;
- kennen die wichtigsten **Assembler-Befehle und deren Argumente**;
- wissen wie der **Stackpointer** manipuliert wird.

Loesungen

Exercise 1 - a)

heron:

...

```

move    $f2, $f12          # $f2 := $f12
move    $3, $0             # $3 := 0 (Register $0 ist immer Null)
ble     $5, $0, $Label3   # if ($5 <= $0) then goto $Label3
li      $f4, 5.0e-1       # $f4 := 0.5

```

\$Label5:

```

div     $f0, $f12, $f2    # $f0 := $f12 / $f2
add     $f0, $f2, $f0     # $f0 := $f0 + $f2
mul     $f0, $f0, $f4     # $f0 := f0 * $f4
???     # ==> Hier fehlt eine Zeile
slt     $2, $3, $5        # if ($3 < $5) then $2 := 1
                                # else $2 := 0

move    $f2, $f0          # $f2 := $f0
bne     $2, $0, $Label5   # if ($2 <> $0) then goto $Label5

```

```

addi   $3, $3, 1
addi   $5, $5, -1

```

Exercise 1 - b)

```
float heron(float a, int n)
{
    if (n > 0) {
        float prev_res = heron(a, n-1);
        return (0.5 * (prev_res + a / prev_res));
    }
    else
        return a;
}
void main()
{
    float sqrt2 = heron(2,10);
}
```

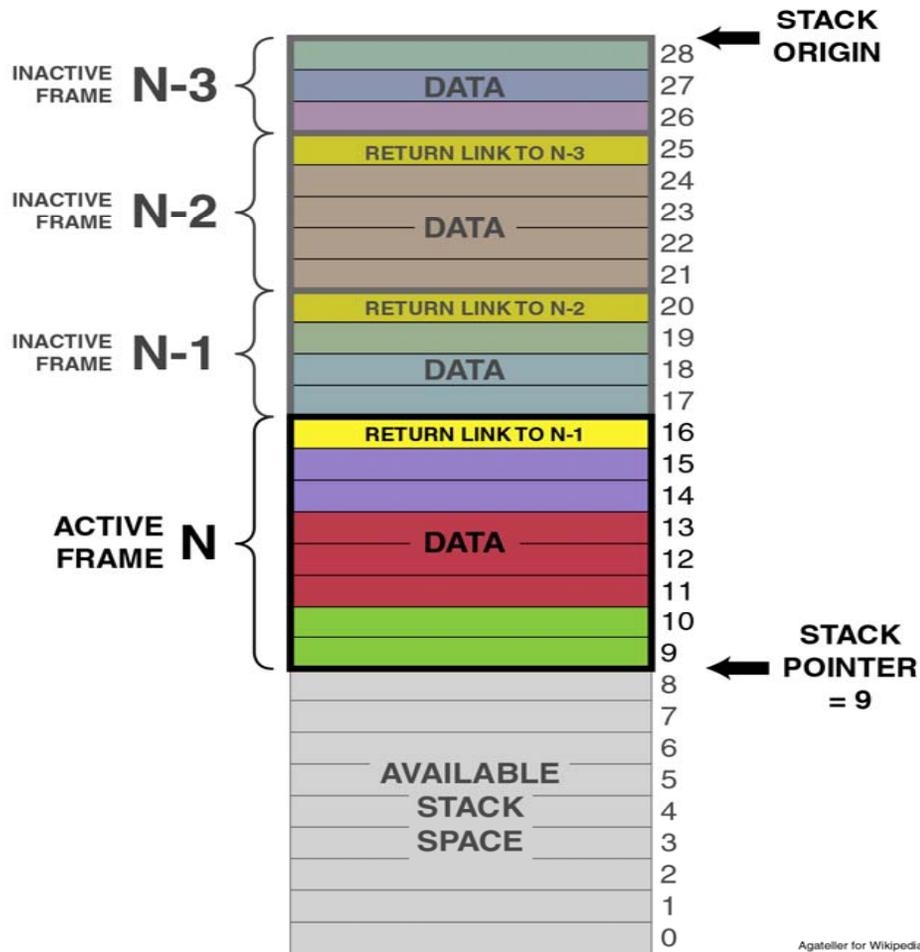
Exercise 1 - c)

Implementierung 1: basiert auf iterativem Verfahren (Schleifen)

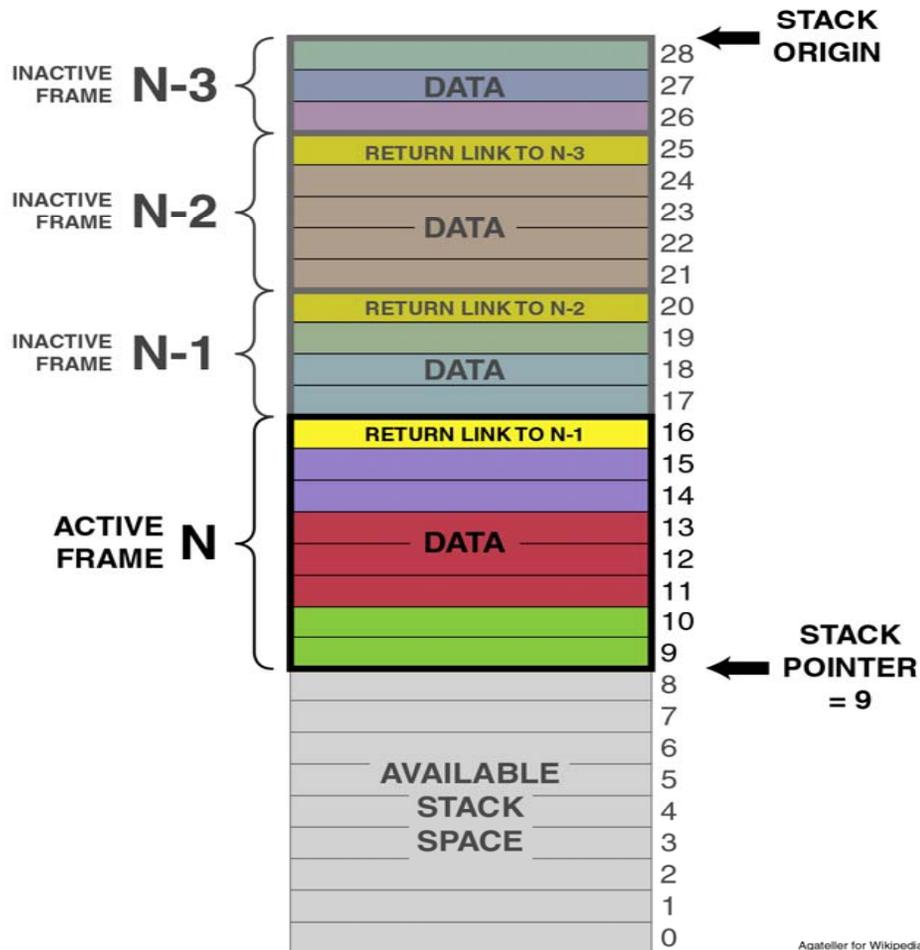
Implementierung 2: basiert auf rekursivem Verfahren (Rekursion)

Schleifen sind im allgemeinen schneller, da sie weniger Instruktionen verwenden und der **Stack auf-/abbau** weniger aufwendig ist.

Exercise 2 – Der Stack



Exercise 2 – Der Stack



Function1:

```

addi $sp, $sp, -12
sw $ra, 4($sp)
jal Function2
lw $ra, 4($sp)
addi $sp, $sp, 12

```

Function 2:

```

addi $sp, $sp, -20
sw $ra, 4($sp)
jal Function3
lw $ra, 4($sp)
addi $sp, $sp, 20

```

Function3:

```

addi $sp, $sp, -16
...
addi $sp, $sp, 16

```

Exercise 2 - Vorbereitungsphase

```
# Vorbereitungsphase:
addi $sp, $sp, -4      # Stack Pointer herabsetzen
sw   $ra, 0($sp)      # Da wir mit mehrfach verschachtelten
                        # Subroutinen arbeiten, müssen wir
                        # die Return Adresse ra auf dem
                        # Stack sichern.

addi $sp, $sp, -4      # Stack Pointer herabsetzen
sw   $fp, 0($sp)      # Alten Frame Pointer auf Stack retten

addi $fp, $sp, 8       # Neuen Frame Pointer setzen
                        # (zeigt auf Argument von Fib)

lw   $t0, 0($fp)      # Das Argument von Fib in t0 laden
li   $t1, 2
bgt  $t0, $t1, rekursion # if n > 2 then goto $rekursion
li   $t0, 1           # else fib(2) = fib(1) = 1
b    abbauphase      # Zur Abbauphase springen
```

Exercise 2 - Rekursion

```
rekursion:
    addi $t0, $t0, -1      # $t0 = n-1
    sw   $t0, 0($sp)      # Argument n-1 auf dem Stack übergeben
    addi $sp, $sp, -4     # Stack Pointer herabsetzen
    jal  Fib              # Fib(n-1)
                          # Das Resultat von Fib(n-1) belassen
                          # wir vorerst noch auf dem Stack
    lw   $t0, 0($fp)      # Argument von Fib wieder holen
                          # t0 = n
    addi $t0, $t0, -2     # t0 = n-2
    sw   $t0, 0($sp)      # push argument n-2 on stack
    addi $sp, $sp, -4     # Stack Pointer herabsetzen
    jal  Fib              # Fib(n-2)

    addi $sp, $sp, 4      # Stack Pointer heraufsetzen
    lw   $t0, 0($sp)      # Resultat von Fib(n-2) vom Stack
                          # holen und in t0 laden
    addi $sp, $sp, 4      # Stack Pointer heraufsetzen
    lw   $t1, 0($sp)      # Resultat von Fib(n-1) vom Stack
                          # holen und in t1 laden
    add  $t0, $t0, $t1    # t0 = Fib(n-2) + Fib(n-1)
```

Exercise 2 - Abbauphase

```
abbauphase:
    addi $sp, $sp, 4      # Im Register t0 liegt das Resultat
    lw   $fp, 0($sp)     # Stack Pointer heraufsetzen
                                # Alter Frame Pointer holen und gleich
                                #   in fp laden
    addi $sp, $sp, 4      # Stack Pointer heraufsetzen

    lw   $ra, 0($sp)     # Return Adresse vom Stack holen
    addi $sp, $sp, 4      # Stack Pointer heraufsetzen
                                # Das Argument von Fib brauchen wir
                                #   nicht mehr.
    sw   $t0, 0($sp)     # Resultat Fib(n) auf dem Stack
                                #   übergeben
```