Übersetzungsfunktionen

1. Anweisungen:

code e; ρ = code$_R$ e ρ
pop

code $(s\ ss)$ ρ = code s ρ
code ss ρ

code ε ρ = ε //leere Befehlsfolge

code $(if\ (e)\ s\ )$ ρ =
code$_R$ e ρ
jumpz A
code s ρ
A : ...

code $(if\ (e)\ s_1\ else\ s_2)$ ρ =
code$_R$ e ρ
jumpz A
code s_1 ρ
jump B
A : code s_2 ρ
B : ...

code $(while\ (e)\ s\ )$ ρ =
A: code$_R$ e ρ
jumpz B
code s ρ
jump A
B : ...

code $(for\ (e_1;\ e_2;\ e_3)\ s)$ ρ =
code$_R$ e_1 ρ
pop
A : code$_R$ e_2 ρ
jumpz B
code s ρ
code$_R$ e_3 ρ
pop
jump A
B : ...

code s ρ = //s sei eine switch-Anweisung

code$_R$ e ρ
check 0 k B
C_0: code ss_0 ρ
jump D
...
C_k: code ss_k ρ
jump D
B: jump C_0
...
jump C_k
D : ...

s = switch (e) { //mit $s \equiv check\ 0\ k\ B$

case 0: ss_0 break;
case 1: ss_1 break;
⋮
case $k-1$: ss_{k-1} break;
default: ss_k
}

check 0 k B =

```
dup
loadc 0
geq
jumpz A
dup
loadc k
leq
jumpz A
jumpi B
A: pop
loadc k
jumpi B
```

2. Ausdrücke:

$code_L\ (\ e_1[e_2]\ )\ \rho$ = $code_R\ e_1\ \rho$
$code_R\ e_2\ \rho$
loadc | t | //sei e_1 vom Typ $t[]$
mul
add

$code_L\ (\ e.a\ )\ \rho$ = $code_L\ e\ \rho$
loadc $(\rho(a))$
add

$code_L\ (\ *e\ )\ \rho$ = $code_R\ e\ \rho$

$code_L\ x\ \rho$ = loadc $(\rho(x))$

$code_R\ e\ \rho$ = $code_L\ e\ \rho$ //sei e ein Feld

$code_R\ q\ \rho$ = loadc q //sei q eine Konstante

$code_R\ e\ \rho$ = $code_L\ e\ \rho$
load

$code_R\ (\ \&e\ )\ \rho$ = $code_L\ e\ \rho$

$code_R\ (\ malloc(e)\ )\ \rho$ = $code_R\ e\ \rho$
new

$code_R\ (\ free(e)\ )\ \rho$ = $code_R\ e\ \rho$
pop

$code_R\ (e_1\ =\ e_2\ )\ \rho$ = $code_R\ e_2\ \rho$
$code_L\ e_1\ \rho$
store

$code_R\ (e_1\ \diamond\ e_2\ )\ \rho$ = $code_R\ e_1\ \rho$
$code_R\ e_2\ \rho$
op //sei op der Befehl für den Operator $\diamond$

3. Funktionen:

$\text{code}_R\ g(e_1, \ldots, e_n)\ \rho$ = $\text{code}_R\ e_n\ \rho$
$\ldots$
$\text{code}_R\ e_1\ \rho$
mark
$\text{code}_R\ g\ \rho$
call
slide (m-1) // m sei der Platz für die aktuellen Parameter

$\text{code}_R\ g\ \rho$ = loadc $(\rho(g))$ // g sei ein Funktionsname

$\text{code}_R\ (*e)\ \rho$ = $\text{code}_R\ e\ \rho$ // e sei ein Funktionszeiger

$\text{code}_R\ e\ \rho$ = $\text{code}_L\ e\ \rho$ // für Paremeterübergabe
move k // e sei eine Struktur der Größe k

code $t\ f(specs)\{V\_defs\ ss\}\ \rho$ = $\_f$: enter q // sezen des EP $q = max + k$
alloc k // k Platz für die lokalen Variablen
code $ss\ \rho_f$ // $\rho_f = \rho \oplus \rho_{specs} \oplus \rho_{V\_defs}$
return // verlassen der Funktion

$\text{code}_L\ x\ \rho$ = $\begin{cases} \text{loadc}\ j & tag = G \\ \text{loadrc}\ j & tag = L \end{cases}$

code return e; ρ = $\text{code}_R\ e\ \rho$
storer -3
return

4. Ganze Programme:

code $p\ \emptyset$ =
enter (k+4)
alloc (k+1)
mark
loadc $\_main$
call
slide (k)
halt
$\_f_1$: $code\ F\_def_1\ \rho$
$\vdots$
$\_f_n$: $code\ F\_def_n\ \rho$

Es gelte:

- p sei das Programm besetehend aus $V\_defs\ F\_def_1 \ldots F\_def_n$
- $\emptyset$ sei die leere Adressumgebung
- ρ sei die globale Adressumgebung
- k sei der Platz für die globalen Variablen