Beispiel: Code für $e \equiv x = y - 1$ mit $\rho = \{x \mapsto 4, y \mapsto 7\}$.

Dann liefert $\text{code}_R\ e\ \rho$:

```
loadc 4

loadc 7
load

loadc 1
sub

store
```

Optimierungen:

Einführung von Spezialbefehlen für häufige Befehlsfolgen, hier etwa:

```
loada q        =   loadc q
                   load

bla; storea q  =   loadc q; bla
                   store
```

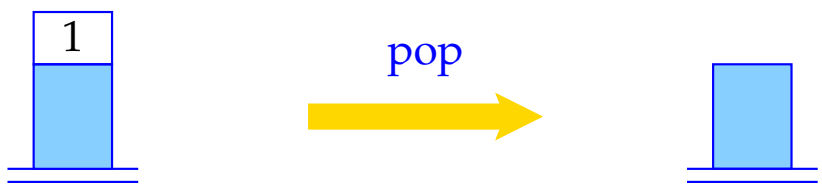
3 Anweisungen und Anweisungsfolgen

Ist e ein Ausdruck, dann ist $e;$ eine Anweisung (Statement).

Anweisungen liefern keinen Wert zurück. Folglich muss der SP vor und nach der Ausführung des erzeugten Codes gleich sein:

$$\text{code } e;\ \rho \quad = \quad \text{code}_R\ e\ \rho$$
$$\text{pop}$$

Die Instruktion pop wirft das oberste Element des Kellers weg ...



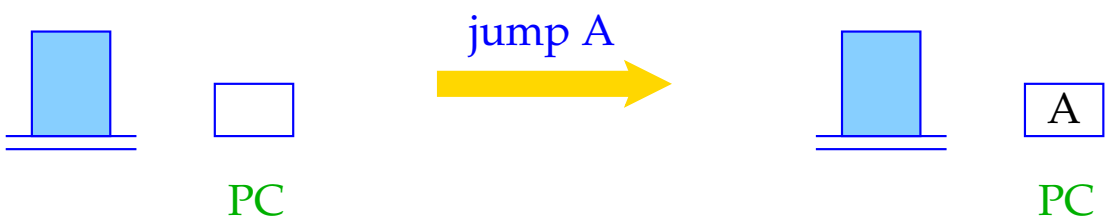
SP--;

Der Code für eine Statement-Folge ist die Konkatenation des Codes for die einzelnen Statements in der Folge:

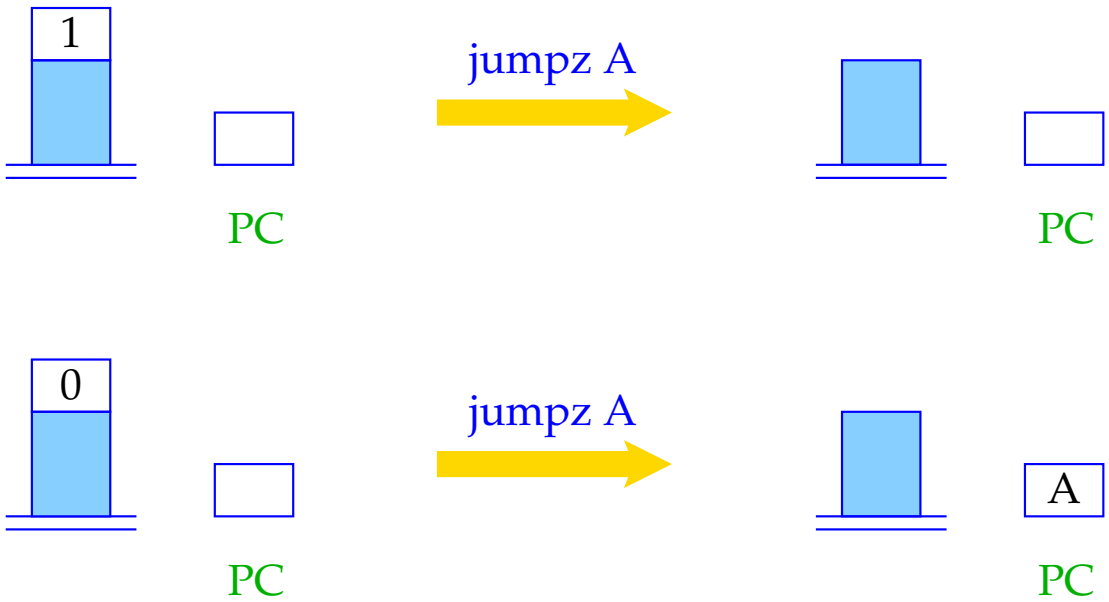
$$\begin{aligned} \text{code}\ (s\ ss)\ \rho &= \text{code}\ s\ \rho \\ &\quad\ \ \text{code}\ ss\ \rho \\ \text{code}\ \varepsilon\ \rho &= \ \ // \quad \textit{leere Folge von Befehlen} \end{aligned}$$

4 Bedingte und iterative Anweisungen

Um von linearer Ausführungsreihenfolge abzuweichen, benötigen wir Sprünge:



PC = A;



if (S[SP] == 0) PC = A;

SP--;

Der Übersichtlichkeit halber gestatten wir die Verwendung von symbolischen Sprungzielen. In einem zweiten Pass können diese dann durch absolute Code-Adressen ersetzt werden.

Statt absoluter Code-Adressen könnte man auch relative Adressen benutzen, d. h. Sprungziele relativ zum aktuellen PC angeben.

Vorteile:

- kleinere Adressen reichen aus;
- der Code wird relokierbar, d. h. kann im Speicher unverändert hin und her geschoben werden.

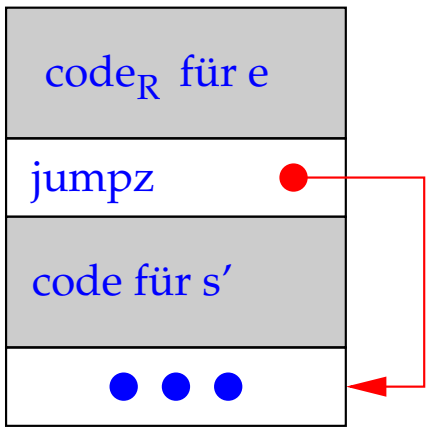
4.1 Bedingte Anweisung, einseitig

Betrachten wir zuerst $s \equiv \textbf{if}\ (e)\ s'$.

Idee:

- Lege den Code zur Auswertung von e und s' hintereinander in den Code-Speicher;
- Dekoriere mit Sprung-Befehlen so, dass ein korrekter Kontroll-Fluss gewährleistet ist!

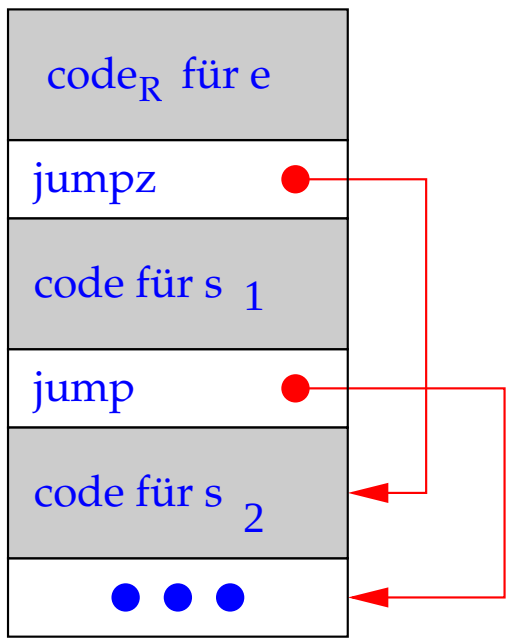
$$
\begin{array}{lcl}
\text{code } s\ \rho & = & \text{code}_R\ e\ \rho \\
 & & \text{jumpz A} \\
 & & \text{code } s'\ \rho \\
 & \text{A}: & \ldots
\end{array}
$$



4.2 Zweiseitiges if

Betrachte nun $s \equiv \textbf{if}\ (e)\ s_1\ \textbf{else}\ s_2$. Die gleiche Strategie liefert:

$$
\begin{array}{rcl}
\text{code } s\ \rho & = & \text{code}_R\ e\ \rho \\
 & & \text{jumpz A} \\
 & & \text{code } s_1\ \rho \\
 & & \text{jump B} \\
\text{A :} & & \text{code } s_2\ \rho \\
\text{B :} & & \ldots
\end{array}
$$



Beispiel: Sei $\rho = \{x \mapsto 4, y \mapsto 7\}$ und

$$
\begin{array}{lll}
s & \equiv & \textbf{if}\,(x > y) \qquad\qquad (i) \\
 & & \quad x = x - y; \qquad\qquad (ii) \\
 & & \textbf{else}\; y = y - x; \qquad\qquad (iii)
\end{array}
$$

Dann liefert code $s\ \rho$:

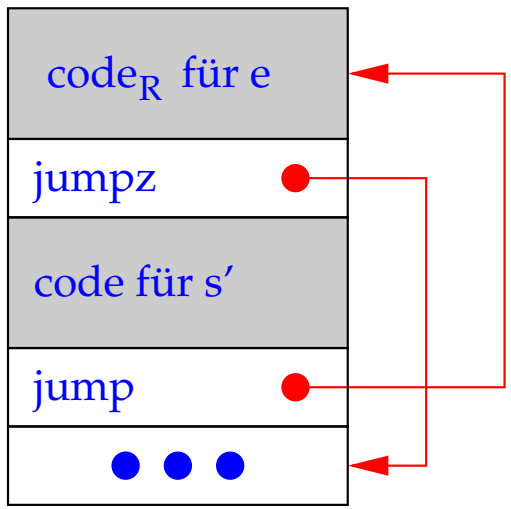
```
loada 4        loada 4        A:  loada 7
loada 7        loada 7            loada 4
gr             sub                sub
jumpz A        storea 4           storea 7
               pop                pop
               jump B         B:  ...
```

(i) (ii) (iii)

4.3 while-Schleifen

Betrachte schließlich die Schleife $s \equiv \textbf{while}\ (e)\ s'$. Dafür erzeugen wir:

$$
\begin{array}{lll}
\text{code } s\ \rho & = & \\
 & \text{A}: & \text{code}_\text{R}\ e\ \rho \\
 & & \text{jumpz B} \\
 & & \text{code } s'\ \rho \\
 & & \text{jump A} \\
 & \text{B}: & \ldots
\end{array}
$$



Beispiel: Sei $\rho = \{a \mapsto 7, b \mapsto 8, c \mapsto 9\}$ und s das Statement:

$$\textbf{while}\ (a > 0)\ \{c = c + 1;\ a = a - b;\}$$

Dann liefert code $s\ \rho$ die Folge:

```
A:  loada 7      loada 9      loada 7      B:  ...
    loadc 0      loadc 1      loada 8
    gr           add          sub
    jumpz B      storea 9     storea 7
                 pop          pop
                              jump A
```

4.4 for-Schleifen

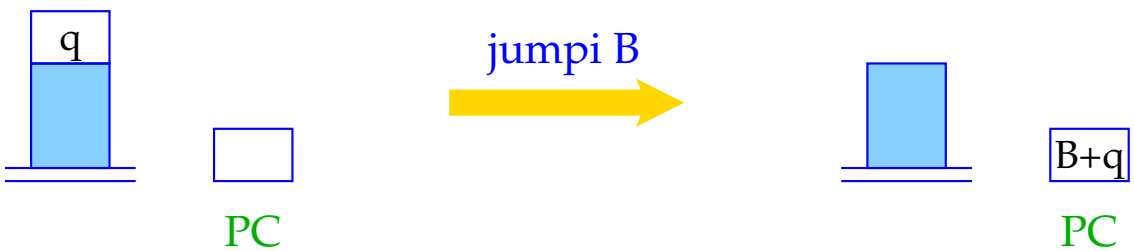
Die **for**-Schleife $s \equiv \textbf{for}\ (e_1; e_2; e_3)\ s'$ ist äquivalent zu der Statementfolge $e_1;\ \textbf{while}\ (e_2)\ \{s'\ e_3;\}$ – sofern s' keine **continue**-Anweisung enthält.
Darum übersetzen wir:

$$
\begin{array}{rcl}
\text{code}\ s\ \rho & = & \text{code}_\text{R}\ e_1 \\
 & & \text{pop} \\
 & \text{A}: & \text{code}_\text{R}\ e_2\ \rho \\
 & & \text{jumpz B} \\
 & & \text{code}\ s'\ \rho \\
 & & \text{code}_\text{R}\ e_3\ \rho \\
 & & \text{pop} \\
 & & \text{jump A} \\
 & \text{B}: & \ldots
\end{array}
$$

4.5 Das switch-Statement

Idee:

- Unterstütze Mehrfachverzweigung in konstanter Zeit!
- Benutze Sprungtabelle, die an der i-ten Stelle den Sprung an den Anfang der i-tem Alternative enthält.
- Eine Möglichkeit zur Realisierung besteht in der Einführung von indizierten Sprüngen.



PC = B + S[SP];

SP--;

Vereinfachung:

Wir betrachten nur **switch**-Statements der folgenden Form:

$$
\begin{array}{lcl}
s & \equiv & \textbf{switch}\ (e)\ \{ \\
 & & \quad \textbf{case}\ 0\text{:} \quad ss_0\ \textbf{break;} \\
 & & \quad \textbf{case}\ 1\text{:} \quad ss_1\ \textbf{break;} \\
 & & \quad \vdots \\
 & & \quad \textbf{case}\ k-1\text{:} \quad ss_{k-1}\ \textbf{break;} \\
 & & \quad \textbf{default:} \quad ss_k \\
 & & \}
\end{array}
$$

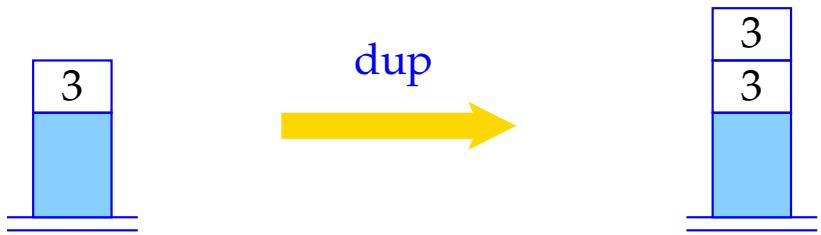
Dann ergibt sich für s die Instruktionsfolge:

$$
\begin{array}{lcllll}
\text{code } s\ \rho & = & \text{code}_R\ e\ \rho & C_0: & \text{code } ss_0\ \rho & B: \quad \text{jump } C_0 \\
 & & \text{check } 0\ k\ B & & \text{jump D} & \ldots \\
 & & & & \ldots & \text{jump } C_k \\
 & & & C_k: & \text{code } ss_k\ \rho & D: \quad \ldots \\
 & & & & \text{jump D} &
\end{array}
$$

- Das Macro check $0\ k\ B$ überprüft, ob der R-Wert von e im Intervall $[0, k]$ liegt, und führt einen indizierten Sprung in die Tabelle B aus.
- Die Sprungtabelle enthält direkte Sprünge zu den jeweiligen Alternativen.
- Am Ende jeder Alternative steht ein Sprung hinter das **switch**-Statement.

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

- Weil der R-Wert von e noch zur Indizierung benötigt wird, muss er vor jedem Vergleich kopiert werden.
- Dazu dient der Befehl dup.
- Ist der R-Wert von e kleiner als 0 oder größer als k, ersetzen wir ihn vor dem indizierten Sprung durch k.



S[SP+1] = S[SP];

SP++;

Achtung:

- Die Sprung-Tabelle könnte genauso gut direkt hinter dem Macro `check` liegen. Dadurch spart man ein paar unbedingte Sprünge, muss aber evt. das **switch**-Statement zweimal durchsuchen.
- Beginnt die Tabelle mit u statt mit 0, müssen wir den R-Wert von e um u vermindern, bevor wir ihn als Index benutzen.
- Sind sämtliche möglichen Werte von e sicher im Intervall $[0, k]$, können wir auf `check` verzichten.

5 Speicherbelegung für Variablen

Ziel:

Ordne jeder Variablen x statisch, d. h. zur Übersetzungszeit, eine feste (Relativ-)Adresse $\rho\, x$ zu!

Annahmen:

- Variablen von Basistypen wie **int**, ... erhalten eine Speicherzelle.
- Variablen werden in der Reihenfolge im Speicher abgelegt, wie sie deklariert werden, und zwar ab Adresse 1.

Folglich erhalten wir für die Deklaration $d \equiv t_1\ x_1; \ldots t_k\ x_k;$ (t_i einfach) die Adress-Umgebung ρ mit

$$\rho\, x_i = i, \quad i = 1, \ldots, k$$

5.1 Felder

Beispiel: **int** [11] a;

Das Feld a enthält 11 Elemente und benötigt darum 11 Zellen.
$\rho\, a$ ist die Adresse des Elements $a[0]$.