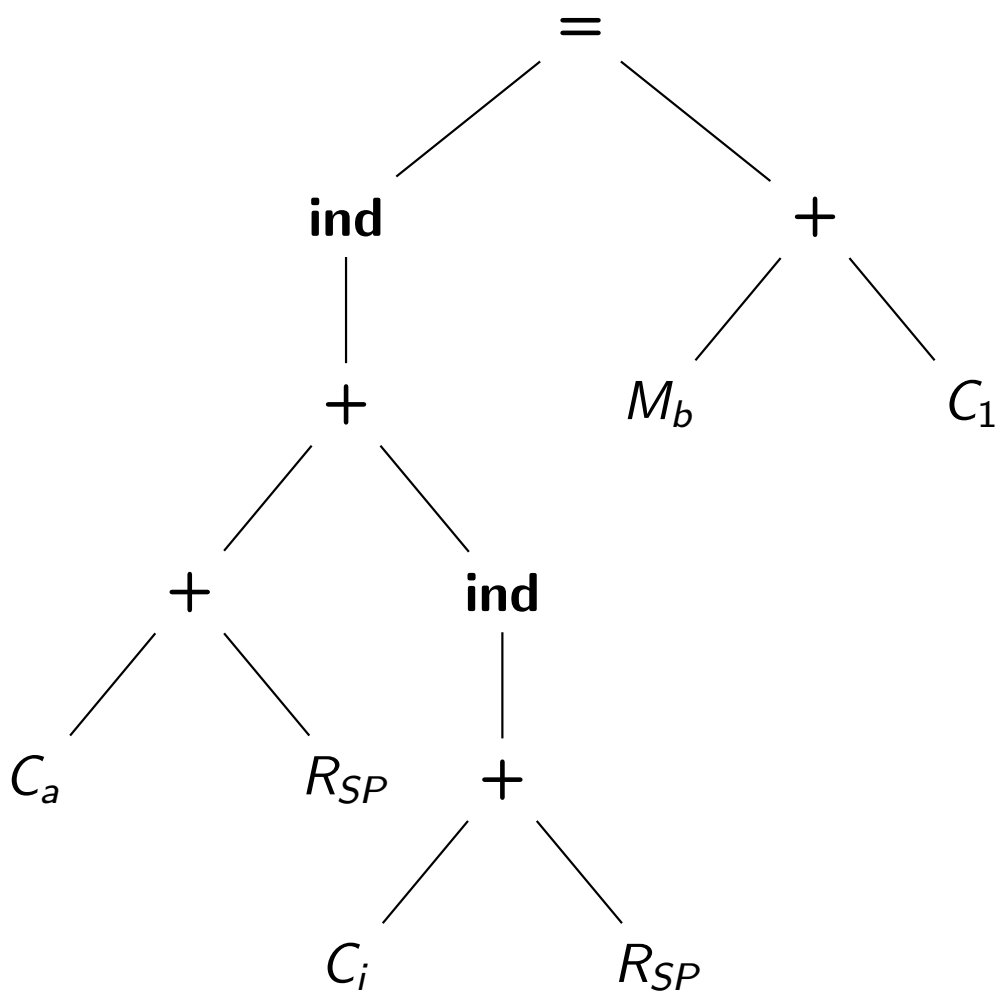
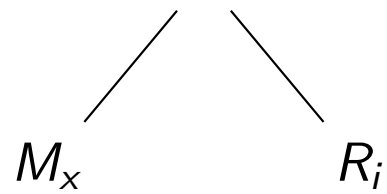
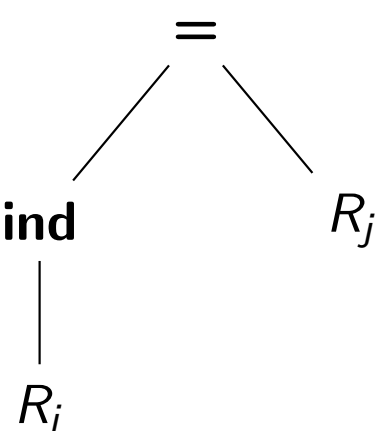
Zwischencodebäume

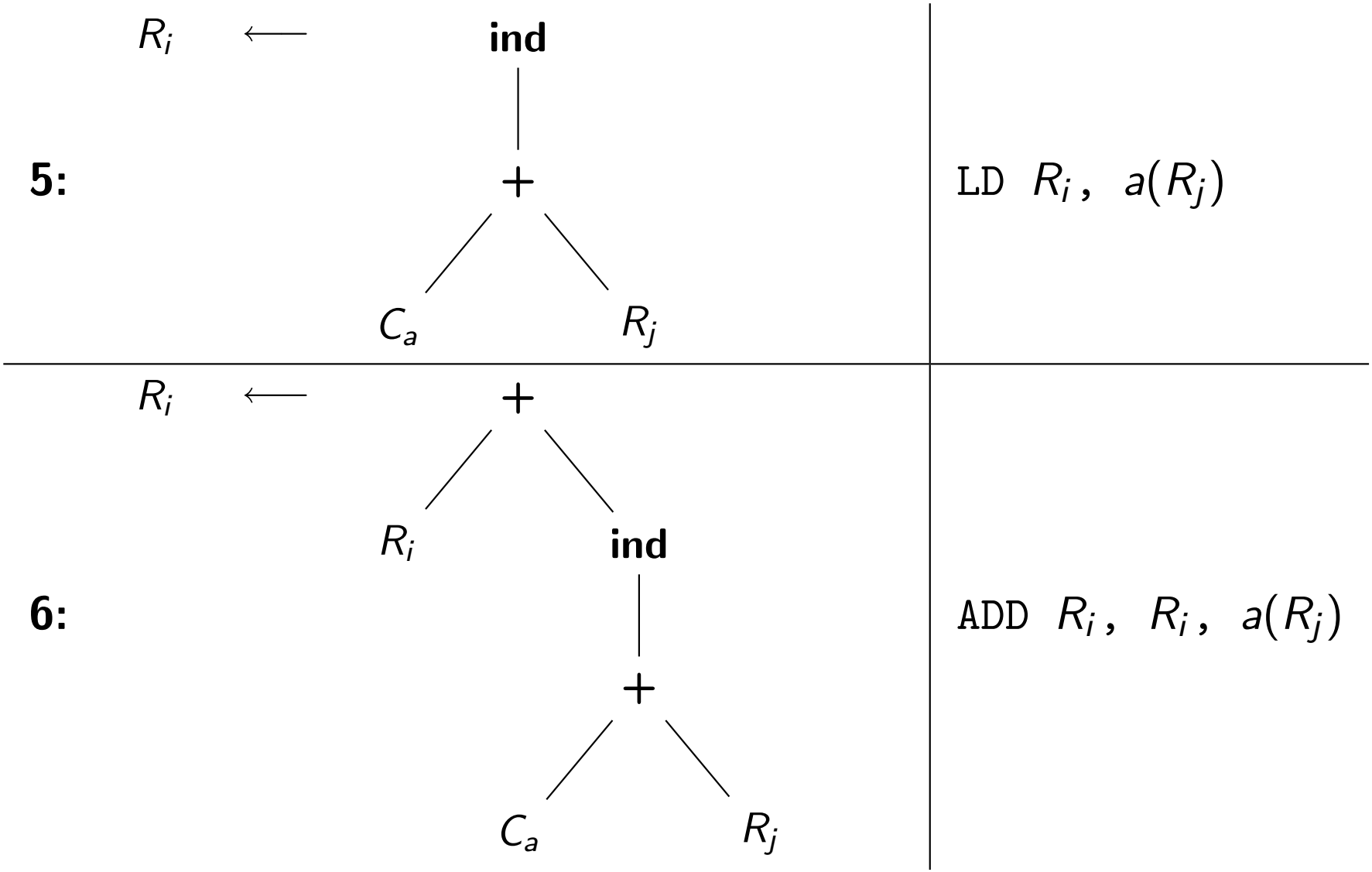


Zwischencodebaum für a[i] = b + 1

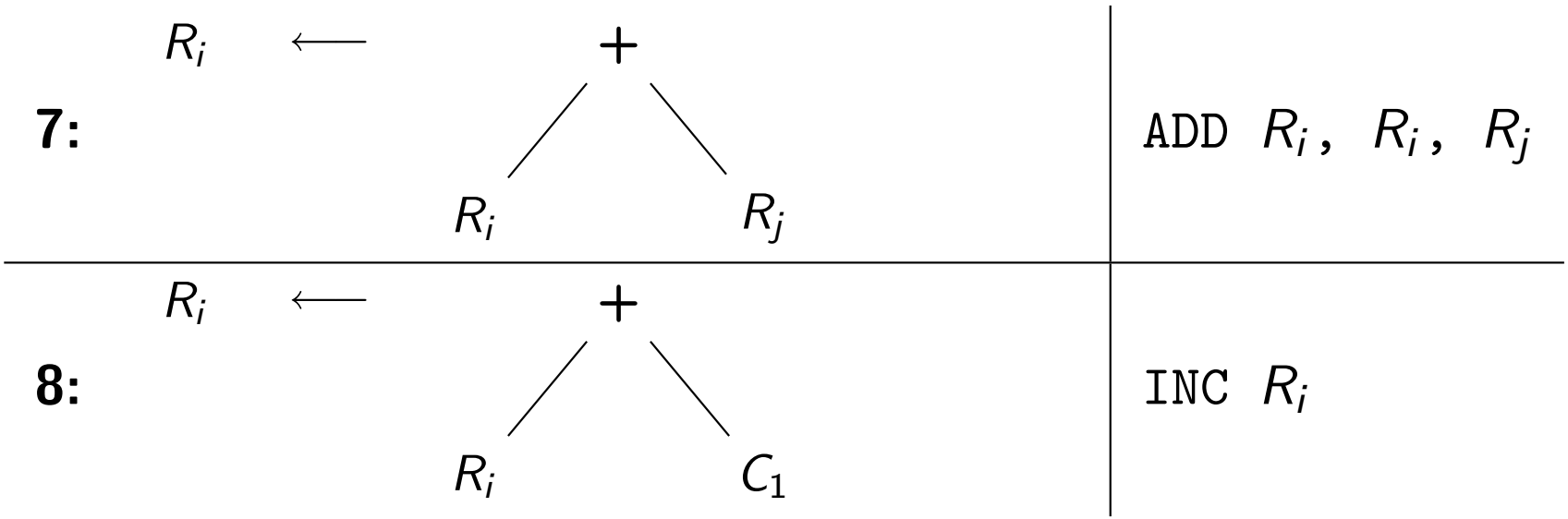
Baumersetzungsregeln (1/3)

1:	R_i	$\longleftarrow$ C_a	`LD` R_i`,` `#a`
2:	R_i	$\longleftarrow$ M_x	`LD` R_i`,` x
3:	M	$\longleftarrow$ 	`ST` x`,` R_i
4:	M	$\longleftarrow$ 	`ST` $*$ R_i`,` R_j

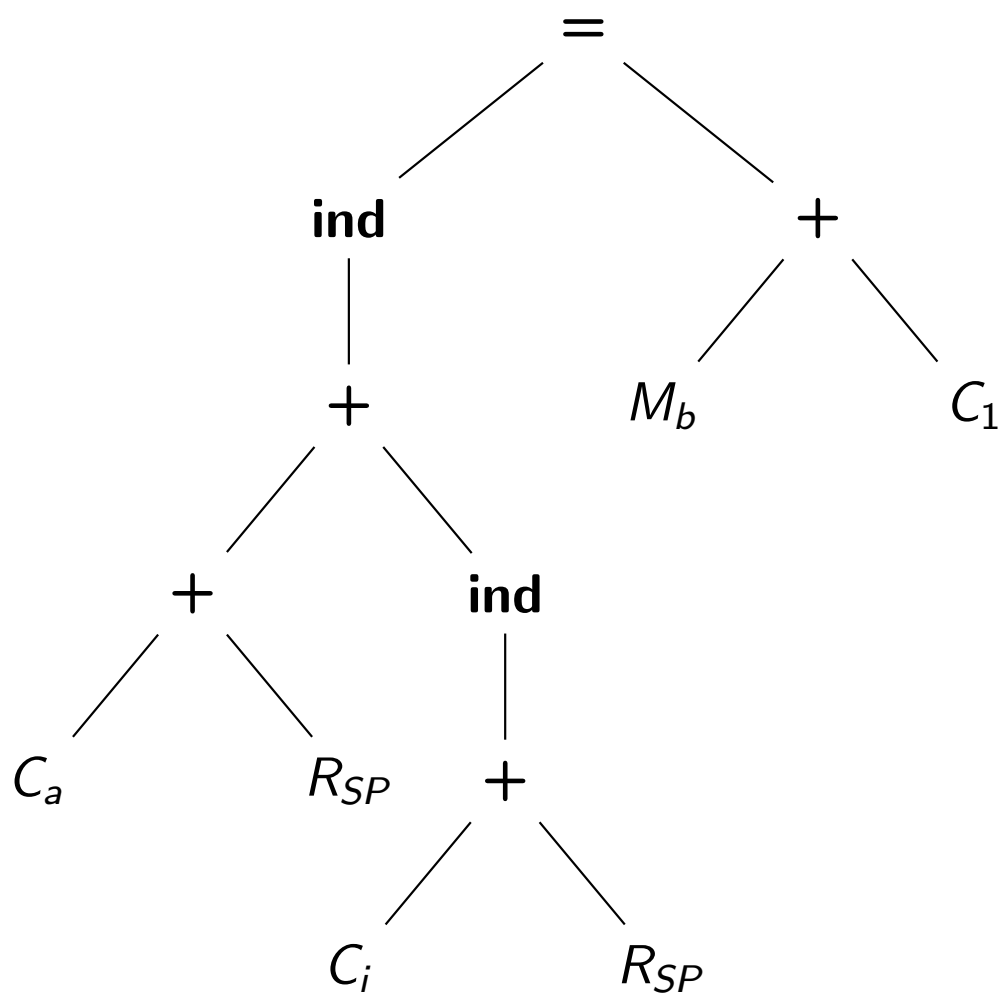
Baumersetzungsregeln (2/3)



Baumersetzungsregeln (3/3)

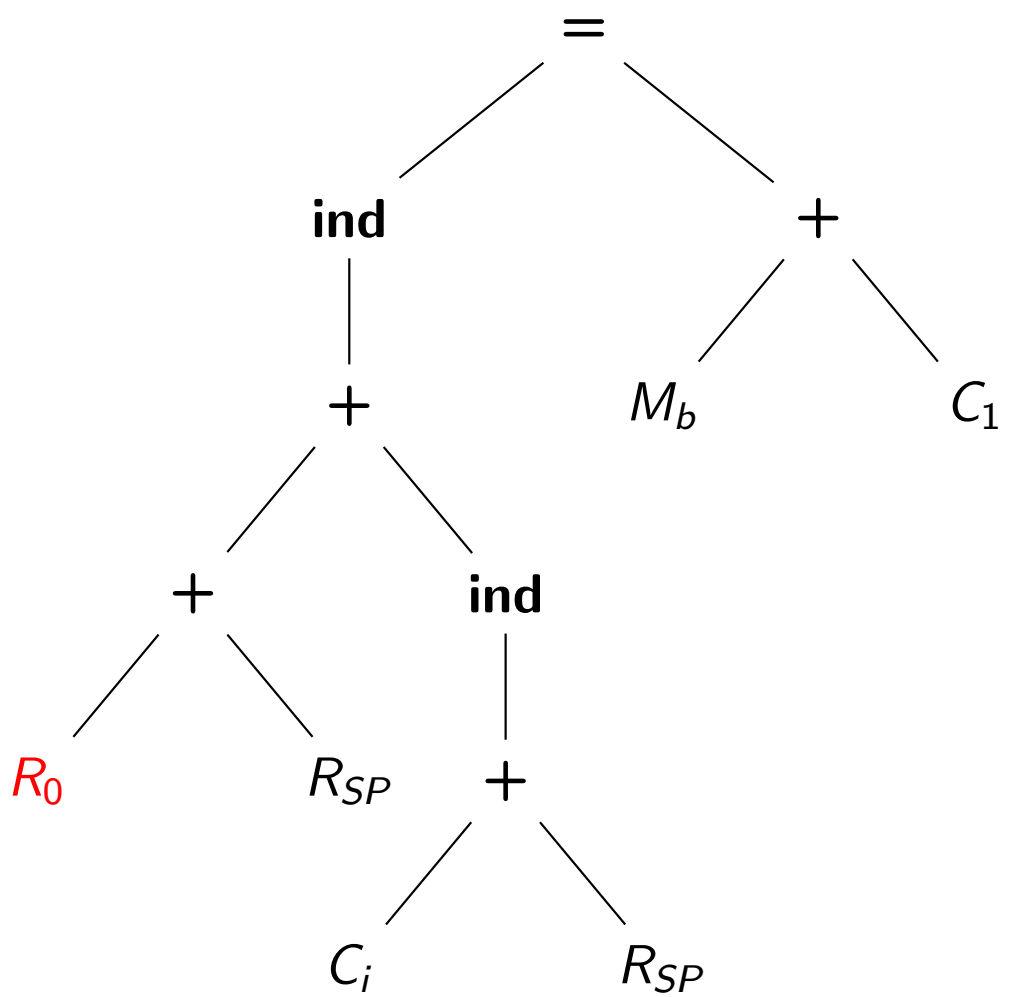


Beispiel Baumübersetzungsverfahren



Anwendung von Regel **1** möglich

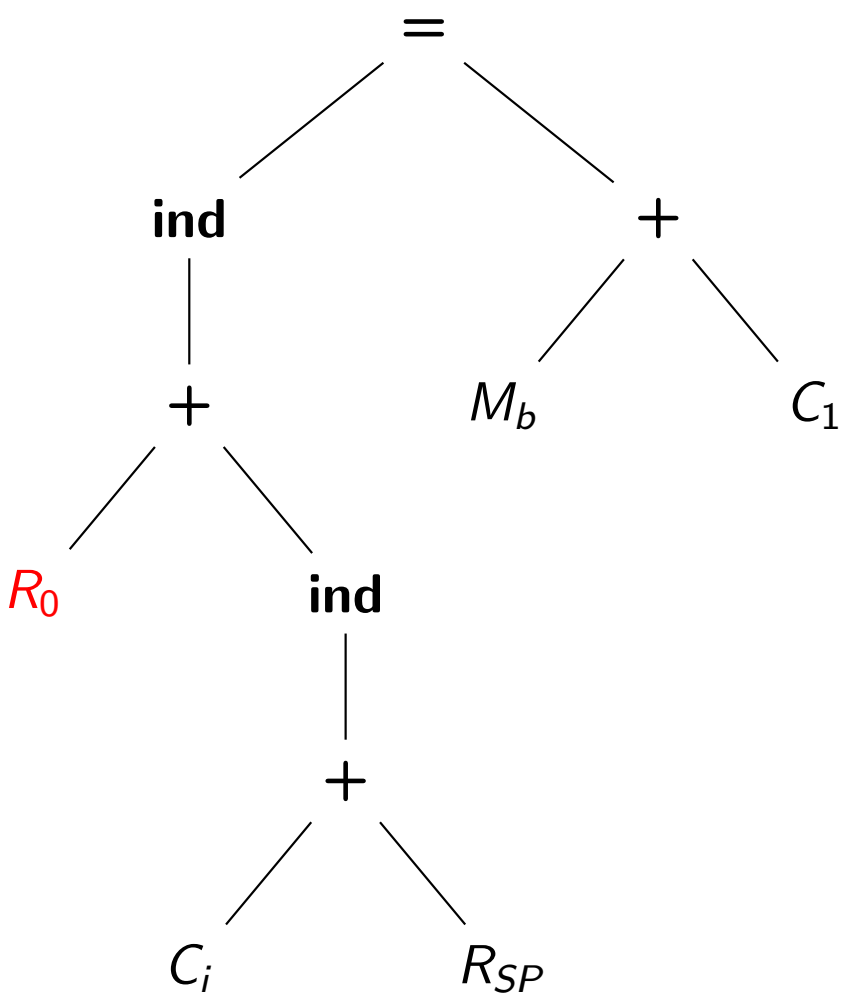
Beispiel Baumübersetzungsverfahren




```
LD R0, #a
```

Anwendung von Regel **7** möglich

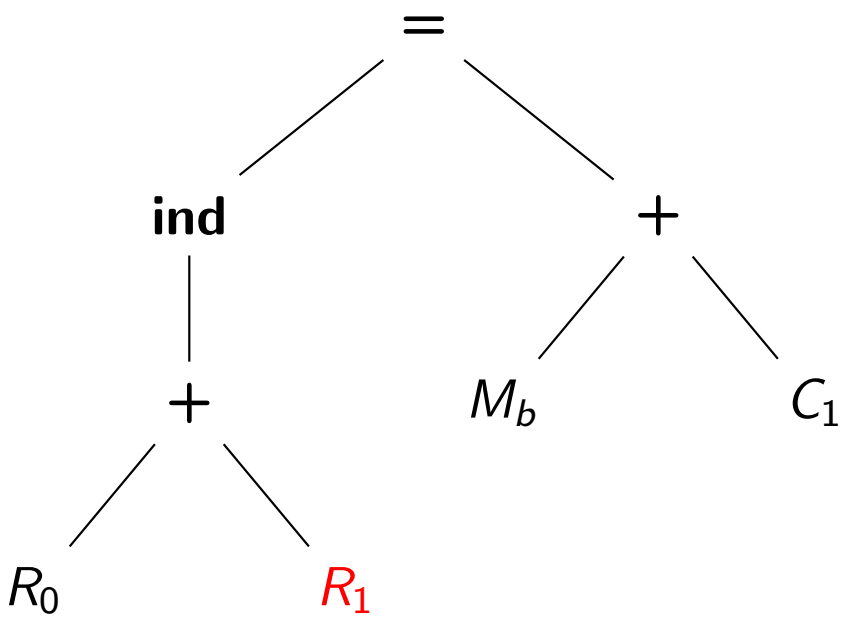
Beispiel Baumübersetzungsverfahren




```
LD R0, #a
ADD R0, R0, RSP
```

Anwendung von Regel **5** möglich

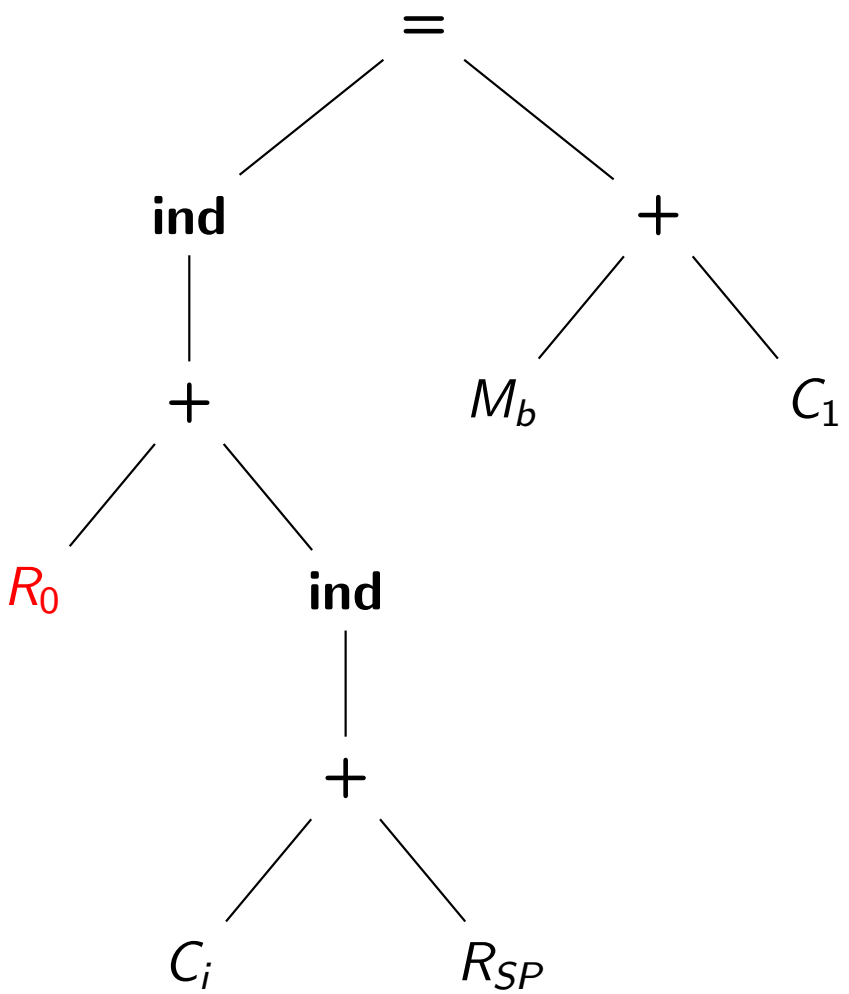
Beispiel Baumübersetzungsverfahren




```
LD R0, #a
ADD R0, R0, RSP
LD R1, i(RSP)
```

Achtung: Es gibt eine Alternative

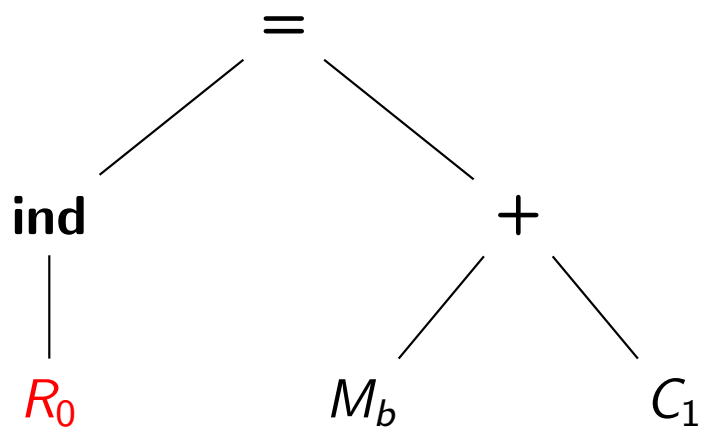
Beispiel Baumübersetzungsverfahren




```
LD R0, #a
ADD R0, R0, RSP
```

Anwendung von Regel **6** möglich

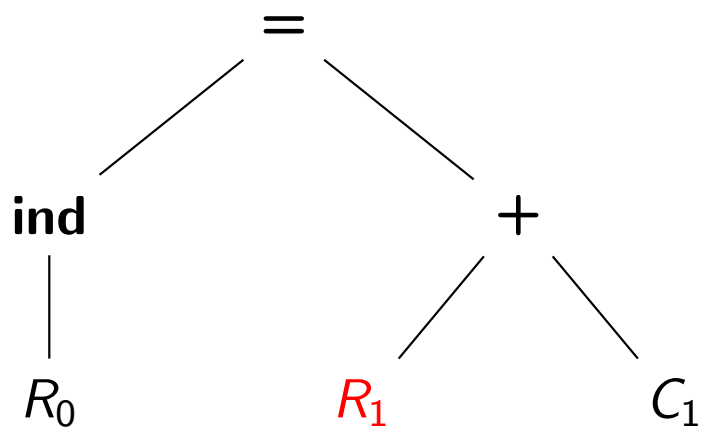
Beispiel Baumübersetzungsverfahren




```
LD R0, #a
ADD R0, R0, RSP
ADD R0, R0, i(RSP)
```

Anwendung von Regel **2** möglich

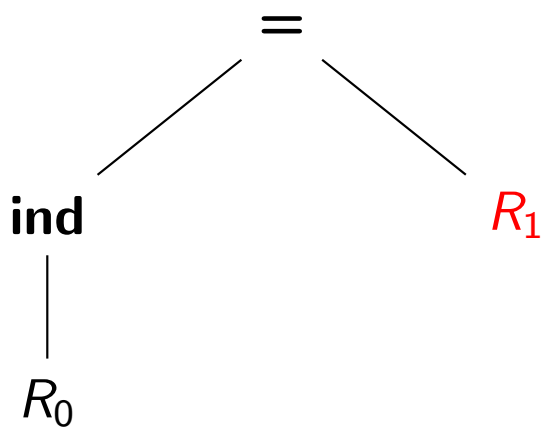
Beispiel Baumübersetzungsverfahren




```
LD  R0, #a
ADD R0, R0, RSP
ADD R0, R0, i(RSP)
LD  R1, b
```

Anwendung von Regel **8** möglich

Beispiel Baumübersetzungsverfahren




```
LD R0, #a
ADD R0, R0, RSP
ADD R0, R0, i(RSP)
LD R1, b
INC R1
```

Anwendung von Regel **4** möglich

Beispiel Baumübersetzungsverfahren

M

```
LD R0, #a
ADD R0, R0, RSP
ADD R0, R0, i(RSP)
LD R1, b
INC R1
ST *R0, R1
```

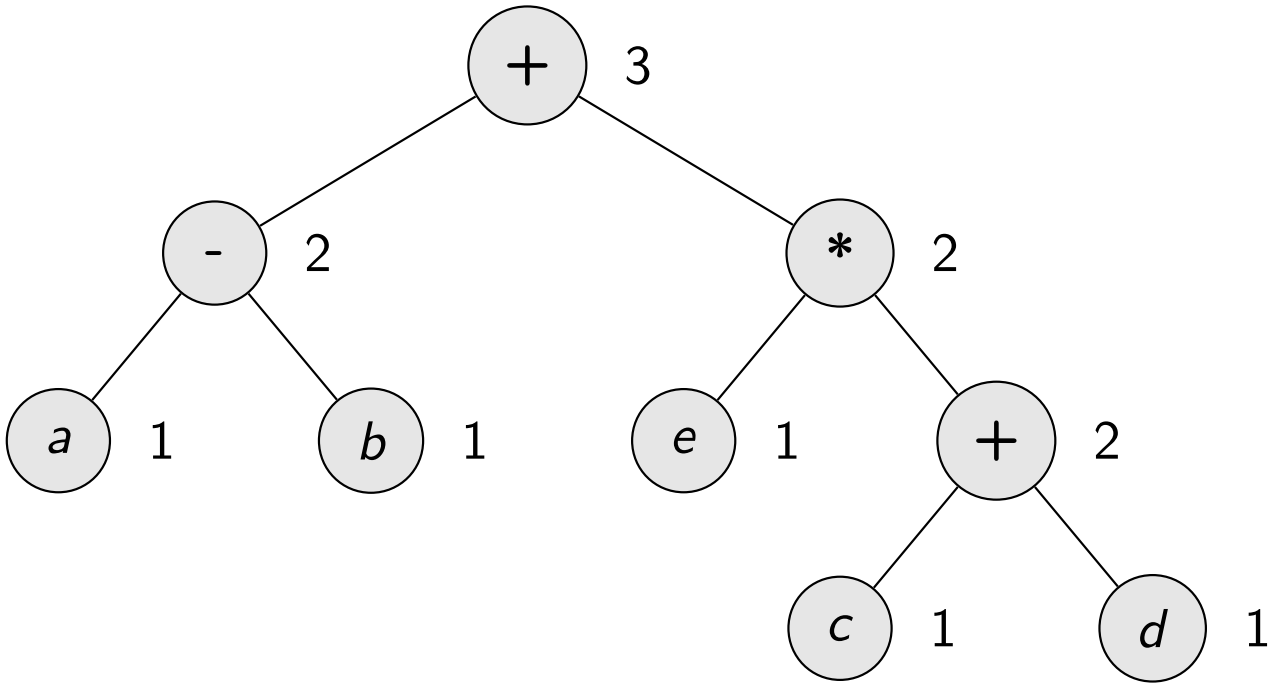
Baumreduktion ist abgeschlossen

Registerverbrauch bei Ausdrücken

Problematik:

- Um eine Operation zu berechnen, berechne zunächst ihre Operanden.
- Wert eines Operanden wird in Register gespeichert; Für weitere Operanden steht ein Register weniger zur verfügung.
- $\Rightarrow$ Optimaler Code berechnet Operanden mit kleinstem Registerverbrauch zuletzt.

Baum mit Ershov-Zahlen



Ausdruck: $(a - b) + e * (c + d)$

Ershov-Zahlen

Ershov-Zahlen geben die Zahl der Register an, die zur Auswertung eines Ausdrucks benötigt werden.
Markieren eines Ausdrucksbaums:

1. Kennzeichne alle Blätter mit 1.
2. Bei 2 Kindern:
   - gleiche Kennzeichnung der Kinder: übernimm Kennzeichnung plus 1
   - sonst: nimm größte Kennzeichnung der Kinder
3. Allgemein: Für absteigend sortierte Markierungen der Kinder $M_1, \ldots, M_n$:

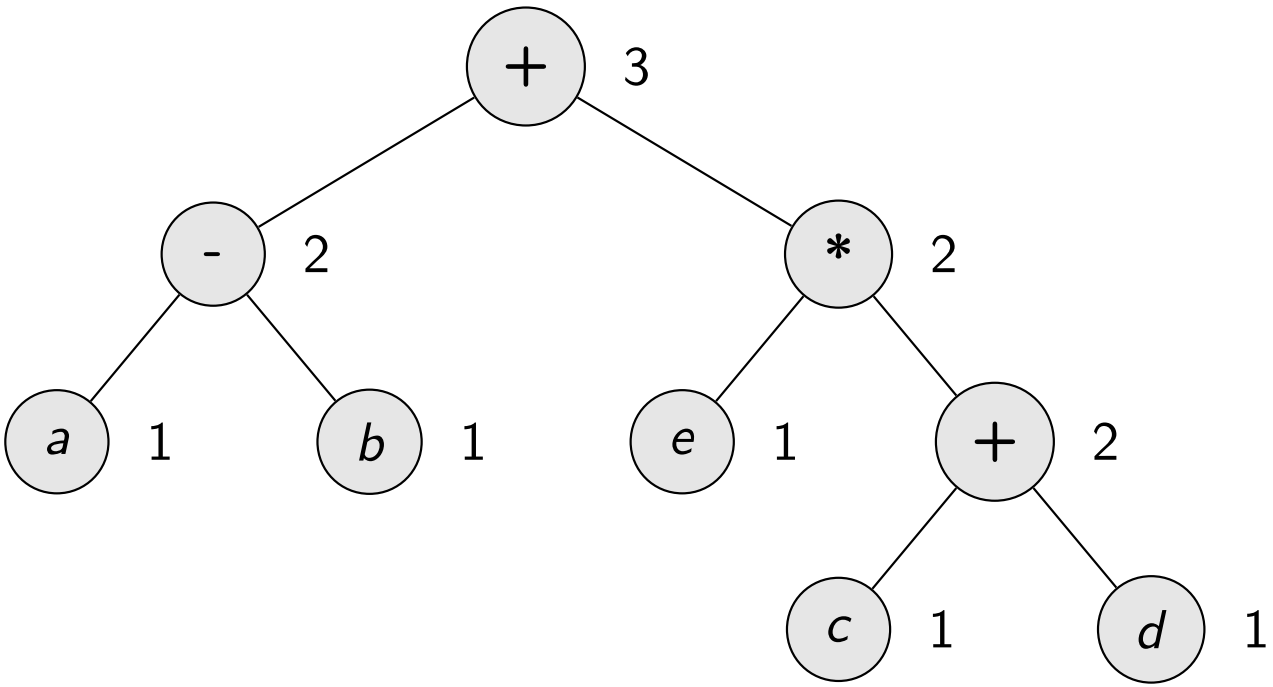
$$\max(M_1, M_2 + 1, \ldots, M_n + (n-1))$$

Codeerzeugung

Mit rekursivem Algorithmus. Register werden relativ zu momentaner Basis b verwendet $(R_b, R_{b+1}, \ldots)$. Beginne an der Wurzel mit Basis $b = 0$. Pro Knoten:

1. Sortiere Kinder nach absteigender Ershov-Zahl (Registerverbrauch).
2. Erzeuge Code für Kinderknoten $K_0, K_1, \ldots, K_{n-1}$ mit Basis $b, b+1, \ldots, b+(n-1)$
3. Erzeuge Operation: OP $R_b, R_b, R_{b+1}, \ldots, R_{b+n-1}$.

Anwendung



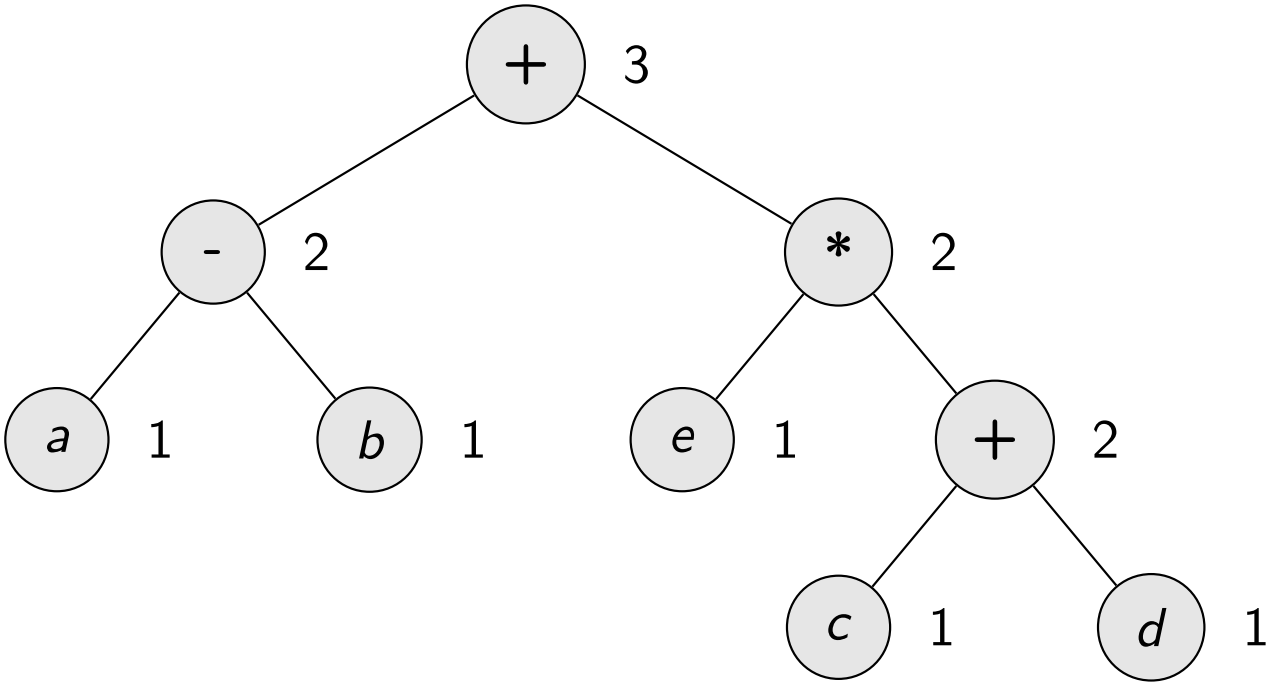

```
LD R1, d
LD R2, c
ADD R1, R2, R1
LD R2, e
MUL R1, R2, R1
LD R2, b
LD R3, a
SUB R2, R3, R2
ADD R1, R2, R1
```


Auslagern

Modifikation bei beschränkter Registerzahl k:

1. Sortiere Kinder nach absteigender Ershov-Zahl (Registerverbrauch).
2. $b_0 \leftarrow b$.
3. Für jeden Kinderknoten $K_i \in \{K_0, \ldots, K_{n-1}\}$.
   a. Erzeuge Code für K_i mit Basis b.
   b. Falls Markierung des nächsten Kindes plus weitere Operanden $M_{i+1} + n - i$ größer als k: Erzeuge Auslagerungsbefehl ST t_x, R_b
   c. $b \leftarrow b + 1$.
4. Lade ausgelagerte Operanden in Register $R_b, R_{b+1}, \ldots$: LD R_b, t.
5. Erzeuge Operation: OP $R_{b_0}, R_{b_0}, R_{b_0+1}, \ldots, R_{b_0+n-1}$.

Anwendung bei 2 Registern




```
LD R1, d
LD R2, c
ADD R1, R2, R1
LD R2, e
MUL R1, R2, R1
ST t1, R1
LD R2, b
LD R3, a
SUB R2, R3, R2
LD R1, t1
ADD R1, R2, R1
```