

# Programmanalysen und Compilerbau

Sommersemester 2005  
Prof. E. Plödereder, E. Wiebe

## 3. Übungsblatt – Lösungsvorschlag

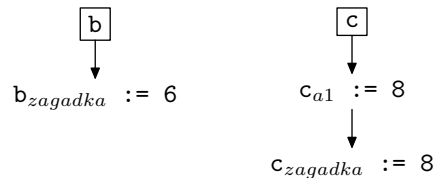
### Aufgabe 3.1

- a. Die Ausgabe des Programms:
- statische Namensbindung: 48 48
  - dynamische Namensbindung: 48 64
- b.
- Symantische Analyse: die relative Adresse von Variablen im Aktivierungsblock ist nicht bekannt.
  - Die Lesbarkeit wird erschwert, da nicht klar ist auf welche Variable zugegriffen wird. Ausserdem besteht keine Garantie, dass lokale Variablen bei Prozeduraufrufen nicht modifiziert werden.
- c. Shallow Access: Globaler Speicherbereich enthält Zugriffslisten für Variablennamen. Zusätzlicher Aufwand bei Prozeduraufrufen sind die `push` und `pop` Operation. Der Zugriff auf den Wert einer Variable bleibt konstant (`top`).

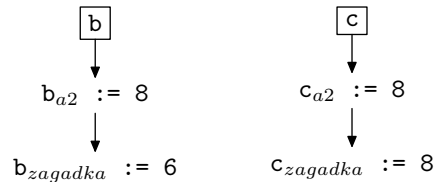
i) Punkt 3



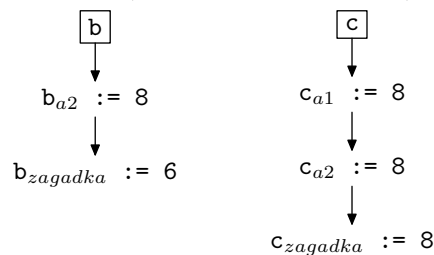
ii) Punkt 1 (a1 vom Hauptprogramm aus ausgerufen)



iii) Punkt 2



iv) Punkt 1 (a1 von a2 ausgerufen)

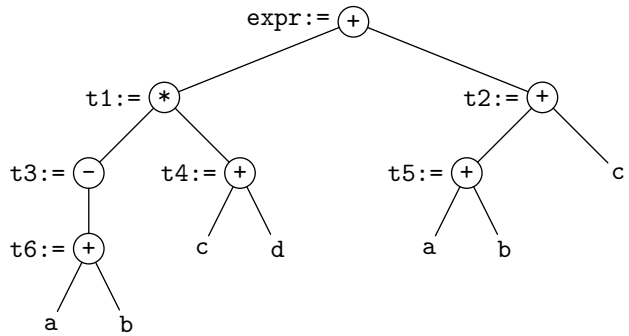


### Aufgabe 3.2

DAG: ähnlich Syntaxbaum, aber gemeinsame Teilausdrücke (-bäume) existieren nur einmal.

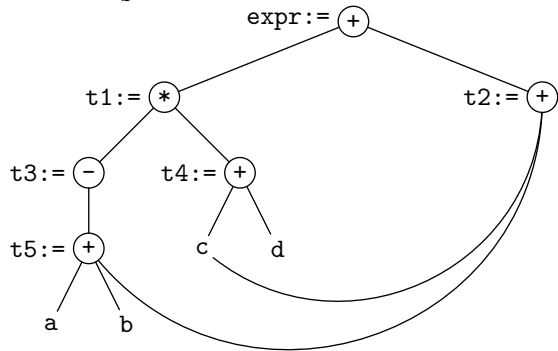
3-Adreß-Code aus Syntaxbaum:

```
t6 := a + b
t3 := -t6
t4 := c + d
t1 := t3 * t4
t5 := a + b
t2 := c + t5
expr := t1 + t2
```



3-Adreß-Code aus DAG:

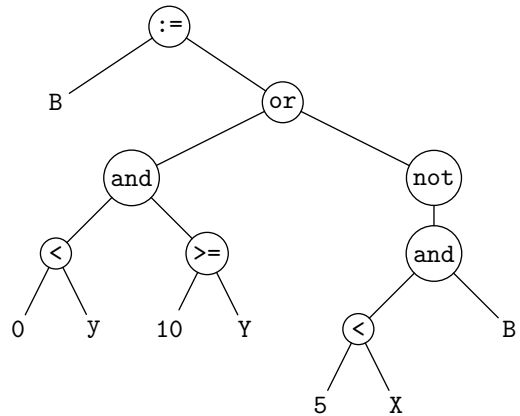
```
t5 := a + b
t4 := c * d
t3 := -t5
t1 := t3 * t4
t2 := t5 + c
expr := t1 + t2
```



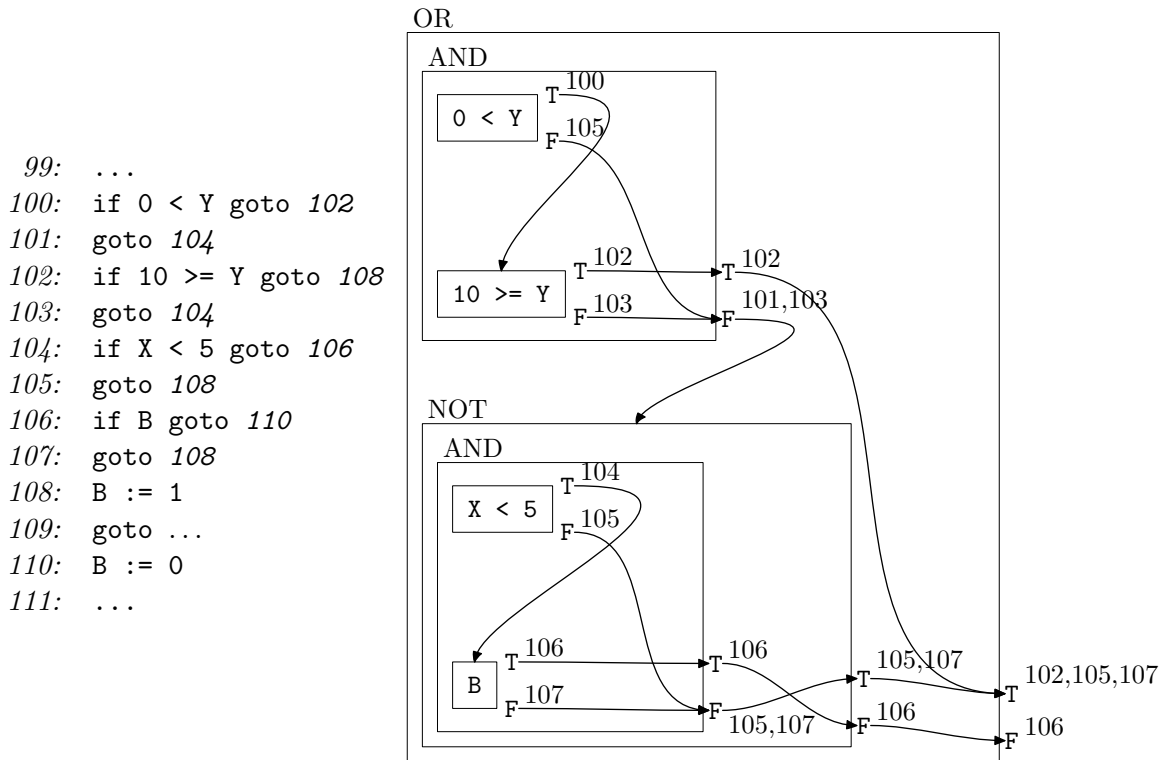
### Aufgabe 3.3

- 3-Adreß-Code aus dem Syntaxbaum:

```
t4 := 0 < Y
t5 := 10 >= Y
t2 := t4 and t5
t7 := X < 5
t6 := t7 and B
t3 := not t6
t1 := t2 and t3
B := t1
```



- Boolesche Operationen durch Verzweigung:



### Aufgabe 3.4

- a. • arithmetische Ausdrücke:
- Auswertung konstanter Ausdrücke zur Übersetzungszeit (constant folding); kann iteriert werden (constant propagation)

```

CONST Pi=3.141; x:= 2*Pi; y:= x*a;
...
IF x > 2 THEN
  p:= 1;
ELSE
  p:= 2;
END;

```
  - Eliminierung gemeinsamer Teilausdrücke
  - Ersetzung „teurer“ Operationen durch billigere („strength reduction“, Beispiel: Multiplikation durch Addition und Shift ersetzen)
  - Ausnutzung algebraischer Eigenschaften: neutrales Element (0 bei + und -, 1 bei \* und /) absorbierendes Element (0 bei \*)
- boolesche Ausdrücke:
- constant propagation und constant folding
  - Eliminierung gemeinsamer Teilausdrücke
  - short circuit evaluation (aber: Seiteneffekte beachten!)
- Unterprogramme:
- Parameterübergabe: Registervergabe, Registerfenster (RISC)
  - textuelle Expansion von Unterprogrammaufrufen (diskutieren: Rekursion, Namensbindungen, mögliche Effizienzverluste durch größeren Code (betrifft auch die Analysierbarkeit))

- bedingte Anweisungen Dead code elimination  
Konstantenoptimierung siehe oben!

b. Von der Zielarchitektur abhängig sind:

- alle Optimierungen, die auf eine möglichst gute Ausnutzung des Befehlssatzes abzielen;
- ferner Optimierungen der Registernutzung bei der Auswertung von Ausdrücken sowie bei der Parameterübergabe;
- strength reduction
- Variable Alignment

c. Bestehen Wechselwirkungen zwischen den verschiedenen Optimierungen?

Ja! Beispiele:

Wenn eine Variable als konstant erkannt wurde, kann dies dazu führen, daß weitere Variablen konstant sind.

Durch die Propagierung von Konstanten kann die Bedingung einer IF-Anweisung konstant werden. Umgekehrt kann die Information, daß eine solche Bedingung konstant ist, dazu führen, daß eine Variable konstant ist. (Beispiele)

Allgemeine Erkenntnis: jedes einzelne Programm wird i.allg. nur von wenigen Optimierungen profitieren. Umgekehrt kann i.allg. jede Optimierungstechnik nur auf wenige Programme angewendet werden. Wenn man optimieren will, sollte man möglichst viele einzelne Optimierungstechniken implementieren, um im Durchschnitt eine brauchbare Verbesserung zu erhalten.