8. Adresszuordnung, Codeerzeugung

- aus Zwischencode *effizienten* Zielcode erzeugen:
- korrekt, hohe Qualität, vorhandene Ressourcen effektiv nutzen
- Zielcode optimal: unentscheidbar

- Einzelheiten von Zielsprache + Betriebssystem abhängig
- typische Probleme für alle:
  - Art der Speicherverwaltung
  - Befehlsauswahl
  - Registervergabe
  - Berechnungsreihenfolge
  - usw.

8. Adresszuordnung, Codeerzeugung

- Input für Codeerzeuger:
  - Informationen der Symboltabelle
  - Zwischencode-Darstellung: 3-Adress-Code, Syntaxbaum, Kellermaschinencode,…
  - Annahme: Eingabe ist lexikalisch., syntaktisch, semantisch korrekt, enthält nötige Typkonvertierungsoperationen, etc.

- Zielcode:
  - absolute Maschinensprache, verschiebbare Maschinensprache, Assembler-Code

8. Adresszuordnung, Codeerzeugung

Problem „Befehlsauswahl“:

- hängt von Befehlssatz der Zielmaschine ab
- Vollständigkeit, Eindeutigkeit
- Ausführungszeiten der Befehle

Beispiel: 3-Adress-Befehl x := y+z

```
MOV y,R0    /* lade y in Register R0 */
ADD z,R0    /* addiere z zu R0 */
MOV R0,x    /* speichere R0 nach x */
```

Aufgabe: Was wird aus

1. a := b+c;
   d := a+e;
2. a := a+1;

8. Adresszuordnung, Codeerzeugung

1. a := b+c;
   d := a+e;

```
MOV b,R0
ADD c,R0
MOV R0,a
MOV a, R0
ADD e, R0
MOV R0 d
```

⟶ nicht effizient: 4. Anweisung überflüssig,
evtl. sogar 3., wenn a nicht mehr benötigt wird

2. a := a+1;

```
MOV a,R0
ADD #1,R0
MOV R0,a
```

⟶ existiert `INC`-Befehl, ist `INC a` effizienter

8. Adresszuordnung, Codeerzeugung

Problem „Registervergabe“:

- Befehle mit Registern als Operanden sind schneller
- → effiziente Nutzung von Registern wichtig
- 2 Phasen:
  - *Registervergabe:* Sammeln von Variablen, die Register belegen sollen
  - *Registerzuweisung:* Auswahl des konkreten Registers pro Variable
- Grund: bestimmte Registerkonventionen der Hardware / des BS

8. Adresszuordnung, Codeerzeugung

Problem „Registervergabe“ (2):

- Beispiel (IBM): Int-Multiplikation, Int-Division nutzt Registerpaare (gerades/ungerades)

  `MUL x,y` x gerades Register, Wert für x steht aber in ungeadem Register, Ergebnis nutzt ganzes Paar

  `DIV x,y` Dividend belegt Registerpaar (gerade/ungerade), x gerades Register, Ergebnis: Rest in geradem, Quotient in ungeradem Register

Aufgabe: kürzeste Assembler-Folgen für

a) t := a + b; t := t * c; t := t / d;

b) t := a + b; t := t + c; t := t / d;

8. Adresszuordnung, Codeerzeugung

Problem „Registervergabe“ (3):

Lösung:

a)
```
MOV a,R1
ADD b,R1
MUL R0,c
DIV R0,d
MOV R1,t
```

b)
```
MOV a,R0
ADD b,R0
ADD c,R0
SRDA R0,32
DIV R0,d
MOV R1,t
```

`SRDA` schiebt Dividenden nach R1, setzt Vorzeichenbits in R0 passend; nötig, da Dividend ganzes Registerpaar belegt

8. Adresszuordnung, Codeerzeugung

Problem „Instruktionskosten“ (1):

- Kosten eines Befehls: 1 + Kosten durch Adressierung
- Adressierung mit Registern: Kosten 0
- Adressierung mit Speicherplätzen / Literalen: Kosten 1
- Beispiele:

`MOV R0,R1`	Kosten 1
`MOV R2,m`	Kosten 2
`ADD #1,R1`	Kosten 2
`SUB 4(R0),*12(R1)`	Kosten 3

- Adresierungen:

absolut:		m
Register:		R
indiziert:	c(R)	c + inhalt(R)
indirekt Register	*R	inhalt(R)
indirekt indiziert	*c(R)	inhalt(c + inthalt(R))

8. Adresszuordnung, Codeerzeugung

Problem „Instruktionskosten“ (3):

Unterschiedliche Codes für 3-Adress-Code a := b + c
(je nach Kontext)

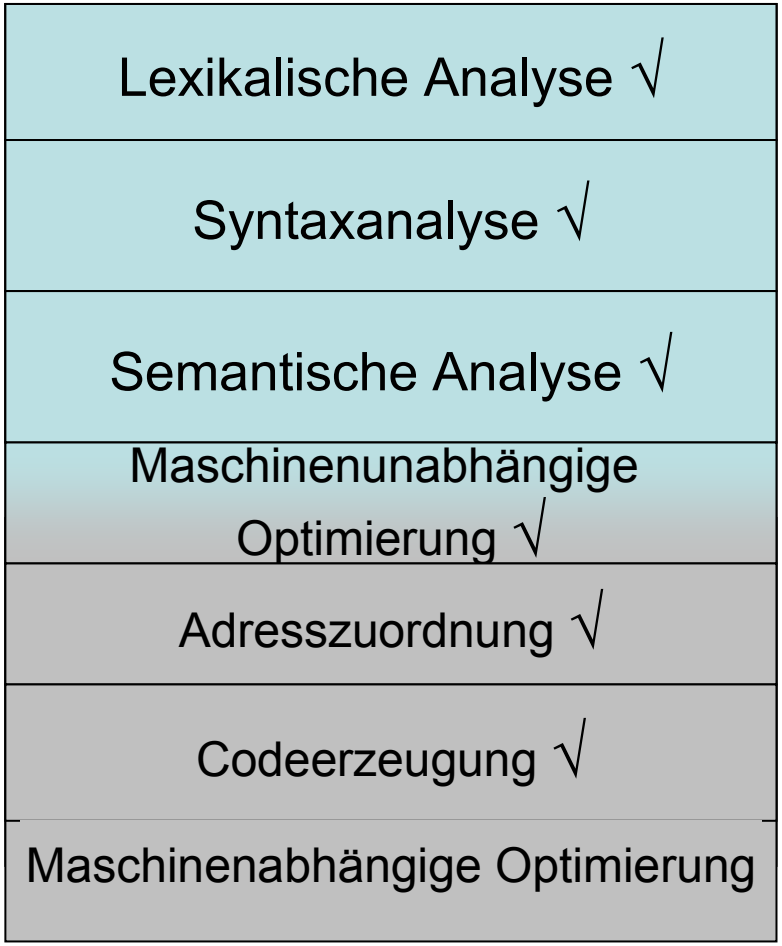
1.
```
MOV b,R0
ADD c,R0
MOV R0,a
```
a,b,c Speicherplätze
Kosten: 6

2.
```
MOV b,a
ADD c,a
```
a,b,c Speicherplätze
Kosten: 6

3.
```
MOV *R1,*R0
ADD *R2,*R0
```
R0,R1,R2 enthalten Adressen von a,b,c, Kosten=2

4.
```
ADD R2,R1
MOV R1,a
```
R1, R2 enthalten Werte von b, c
Wert b wird anschl. nicht mehr benötigt, Kosten = 3

Endstand:


Lexikalische Analyse √
Syntaxanalyse √
Semantische Analyse √
Maschinenunabhängige Optimierung √
Adresszuordnung √
Codeerzeugung √
Maschinenabhängige Optimierung