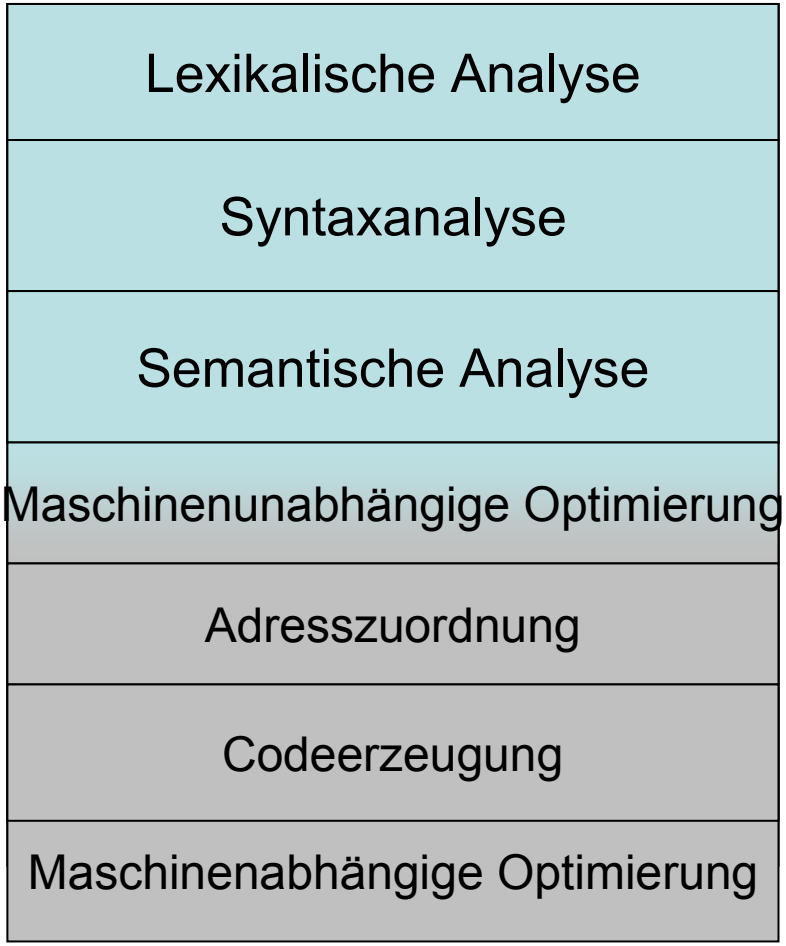
2. Aufbau von Compilern

Aufbau Compiler:



2. Aufbau von Compilern

Maschinenunabhängige Optimierungen/

Indizien für Laufzeitfehler:

- Ausführungspfad nutzt Variablenwert, ohne dass dieser initialisiert ist
- unerreichbare Programmteile
- definierte, aber nie aufgerufene Funktionen
- Programmvariable mit nie verändertem Wert: Ersetzen der Variablen durch Wert (Konstantenpropagation)
- Parameter mit immer gleichem Wert: ersetzen durch Wert

2. Aufbau von Compilern

Weitere Transformationen durch optimierende Übersetzer:

- Herausziehen von schleifeninvarianten Berechnungen aus Schleifen
- Herausziehen von Ausdrücken, die nur Variablen enthalten, die außerhalb einer Funktion gebunden sind, aus der Funktion; stattdessen Übergabe als zusätzlicher formaler Parameter
- Eleminieren von redundanten Berechnungen

2. Aufbau von Compilern

Adresszuordnung:

- abhängig von z.B. Wortlänge, Adresslänge, direkt adressierbaren Einheiten und Existenz von Befehlen zum effizienten Zugriff auf diese
- Zuordnung im Beispiel: Maschine mit Ganzwortadressierung ⇨

[1] ((id(1),(var,real,0)), (id(2),(var,real,2)))

[2] (var,real,0) [3] (var,real,2)

2. Aufbau von Compilern

Code-Erzeugung:

- nutzt Adressierung aus vorherigem Schritt
- daneben Registerzuteilung
- wählt möglichst gute Befehlsfolge pro Anweisung

- Beispiel: var a,b: int; a:=2; b:=a*a+1; ….

Befehle der Zielmaschine:

load	adr, reg
store	reg, adr
loadi	int, reg
addi	int, reg
mul	adr, reg
inc	reg

Übersetzung Beispielprogramm:

```
loadi   2, R1
store   R1, 0
load    0, R1
mul     0, R1
addi    1, R1
store   R1, 1
```

2. Aufbau von Compilern

Maschinenabhängige Codeoptimierung:

- „Peephole“-Optimierer
- eliminiert überflüssige Befehle
- ersetzt allgemeine Befehle in Spezialfällen durch effizientere

- Beispiel:

Optimiertes Beispielprogramm:

```
loadi   2, R1
store   R1, 0
mul     0, R1
inc      R1
store   R1, 1
```

2. Aufbau von Compilern

Konzeptionelle Struktur eines Compilers:

- Folge von Teilprozessen
- Informationsfluss: Einbahnstraße
- Zwischendarstellungen: bekannte Mechanismen der Theorie der formalen Sprachen

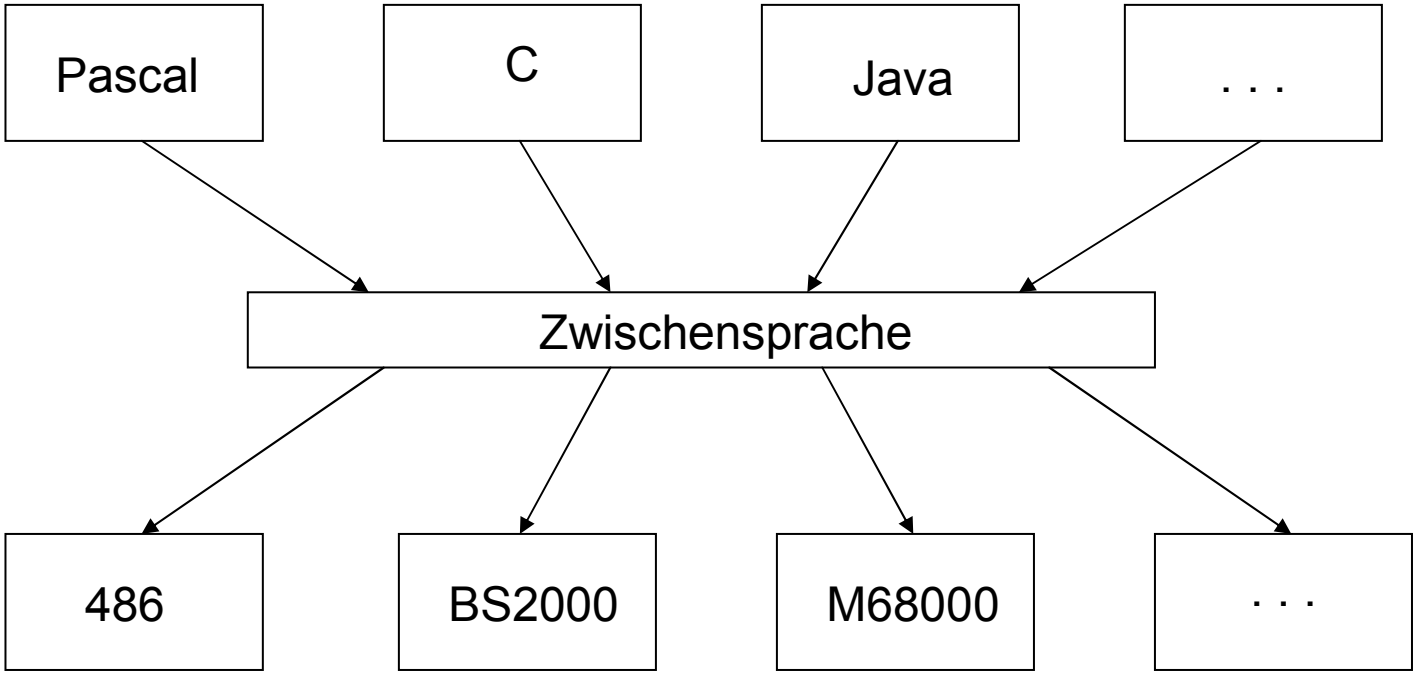
Realer Übersetzer:

- oftmals verschränkte Teilaufgaben
- Zwischendarstellungen stückweise weitergeben
- Hauptprogramm-Unterprogramm-Beziehung zwischen Modulen
- Beispiel: Scanner und Sieber in einem Pass

2. Aufbau von Compilern

Vorteil der Zwischensprache:

Bei *n* Quellsprachen und *m* Maschinen Reduktion auf *n*+*m* Programme



2. Aufbau von Compilern

Fragestellungen zu Kapitel 2:

- Welche Phasen durchläuft ein Compiler?
- Was passiert während jeder einzelnen Phase?
- Was ist eine Symboltabelle? Wann wird sie gefüllt/benutzt?
- Welche Struktur erzeugt die Syntaxanalyse?
- Nennen Sie Beispiele für maschinenunabhängige Optimierungen.
- Wieso ist die Adresszuordnung maschinenabhängig?
- Welche Vorteile hat es, während der Compilierung den Code erst in eine Zwischensprache zu übersetzen?
- Was bringt eine maschinenabhängige Optimierung?