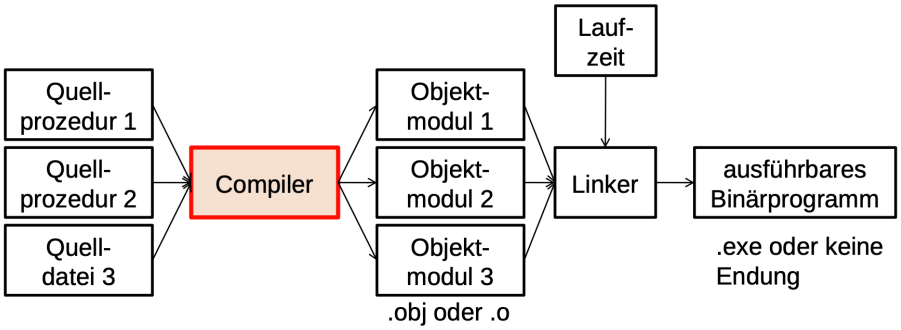
1 Compiler

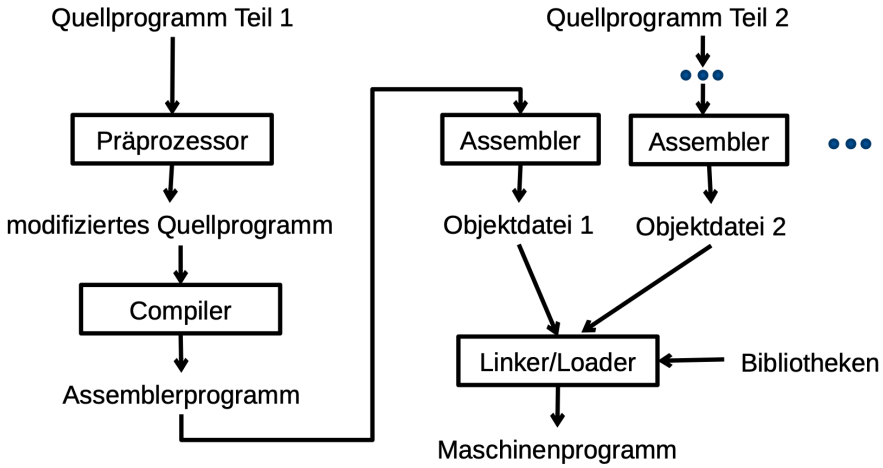
Uebersetzung von Hochsprache nach Assembler/Binaer

- getrennt fuer jede Quelldatei
- normalerweise implizit mit Assembler (”-s” Opflag fuer Ausgabe des Assemblers)
- erzeugt einzelne Objektdateien (.obj)

Objektdateien werden zum ausfuehrbaren Programm gelinkt (oft auch implizit)



Gesamtprozess der Uebersetzung:



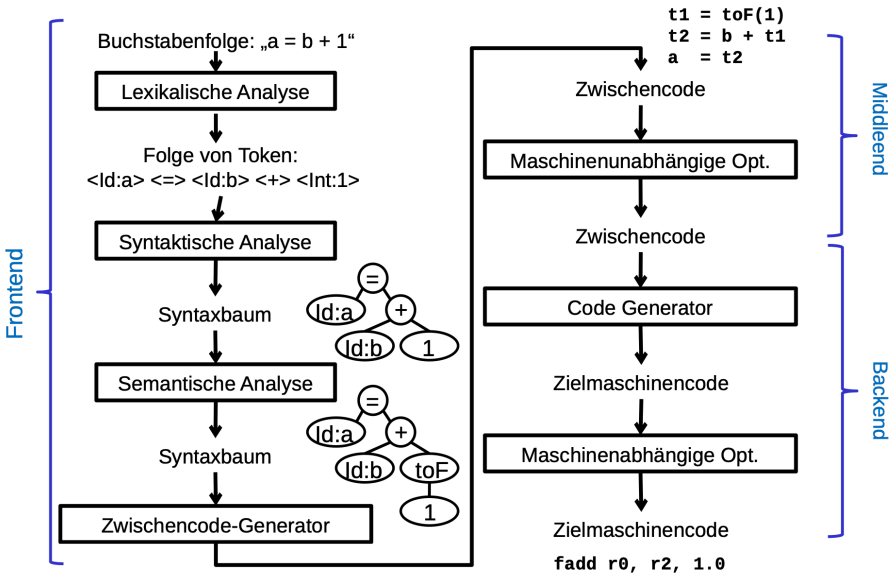
1.1 Praeprozessor

- Eingabe und Ausgabe sind Quelltext
- implizit in C/C++
- ueblichste Variante: C/C++ Makros (texturelle Ersetzung, durch "#")
- Konvention: Grobuchstaben
- alleiniger Praeprozessoraufruf durch "-E" Opflag

Praeprozessor Kommandos:

- #include <file> (Einbindung von Standard-Header Dateien)
- #include "file" (Einbindung von allgemeinen Header Dateinen)
- #define <name> <text> (Makrodefinition zur Textersetzung)
- #define <name>(param$_1$,...) <text> (Makrodefinition mit Parametern)
- #if / #else / #elif / #endif (Ein/Ausblendung von Text unter Bedingungen)

1.2 Compileraufbau



1.3 Assemblierer

Generierung von Binarcode aus Assemblercode:

- direkte Uebersetzung jeder Operation
- Generierung einer Objektdatei (.o tpyischerweise)

Realisierung als Zwei-Pass-Assemblierer (Pass = Iteration ueber Eingabeprogramm):

- Aufloesung von Vorwaertsreferenzen:

    - Marken benutzen bevor sie definiert sind
    - Bsp. Vorwaertssprung

- Code kann erst erzeugt werden wenn die Adresse der Marke bekannt ist

1.3.1 Erster Lauf

- Lesen des Programmtextes
- Erzeugen einer Zwischendarstellung fuer den zweiten Lauf

Instruction Location Counter (ILC):

- bestimmt die Adresse des aktuellen Befehls
- wird nach Verarbeitung des Befehls um die Befehlslaenge erhoeht

Verarbeitung eines Befehls:

- Makros durch ihren Text ersetzen
- Symbole (Marken, Variablen) mit dem aktuellen ILC in die Symboltabelle eintragen
- Inkrementieren des ILC

Verwendet: Makrotabelle, Symboltabelle, Opcode-Tabelle

Opcode-Tabelle:

- Option: Einteilung der Befehle in Befehlsklassen
- Befehle der gleichen Klasse werden bei der Codegenerierung gleich behandelt

Symboltabelle:

- Laenge des Datenfeldes
- Zugriffsrechte (z.B. ob Zugang ausserhalb der Prozedur zugaenglich sein soll)
- Relokationsbits (Art der benoetigten Adresse)

1.3.2 Zweiter Lauf

- Erzeugung des Objektprogramms
- Evtl. Ausgabe des Assemblierlistings
- Bereitstellung von Information fuer den Linker

Fehlererkennung und Behandlung im Assemblierer:

- Moegliche Fehler:

    - Verwendung eines nicht definierten Symbols
    - Mehrfach Definition von Symbolen
    - Opcode nicht zulaessig
    - zu viele, zu wenige Operanden fuer eine Instruktion
    - Register werden falsch verwendet

- Behandlung:

    - Wiederaufsetzen der Verarbeitung
    - Ausgabe einer Fehlermeldung

Ergebnis: Binaer- oder Obj. Datei

1.4 Laufzeitsystem

- liegt auf dem Linker auf
- Speichermanagement
- Zugriff auf Bibliotheken oder andere Objektmodule
- Schnittstelle zu Betriebssystem fuer Ein-/Ausgabe etc.

1.4.1 Speicherverwaltung

Statische Daten:

- Teil der Obj. Dateien
- einzelne Segmente in ausfuehrbaren Dateien
- z.T. vordefinierte Werte
- Verfuegbar ueber ganze Prozesslebenszeit

Stack Daten:

- Funktionsparameter (je nach Konvention)
- lokale Variablen
- Register Spilling
- explizite Allokation $\rightarrow$ begrenzte Lebenszeit

Heap Daten:

- explizite Allokation durch Funktionen des Laufzeitsystems
- malloc / calloc → Speicherplatzfestlegung
- free → Speicherplatzfreigebung

1.5 Linker

Getrennte Uebersetzung von Prozeduren / Gruppen von Prozeduren:

- reduziert Speicherbedarf des Assemblierers / Compilers
- erhoeht die Wartbarkeit des Programms
- Bereitstellung von Bibliotheken

Zusammenbinden einzelner Programmteile

1.6 Struktur eines Objektmoduls

- Unterteilung in Abschnitte
- genaue Struktur haengt vom Objektformat ab → weit verbreitet: ELF

ELF Struktur:

- Plattform uebergreifend
- flexibel
- erweiterbar

1.7 Relokationsproblem

Objektmodule werden in einem virtuellen Adressraum gebildet
→ Startadresse ab 0 (Bereich reserviert fuer OS → Verlagerung)

Konsequenzen:

- Spruenge sind nicht mehr korrekt
- Adressen von Variablen sind nicht mehr korrekt

1.8 Zugriff auf Programmbibliotheken

- koennen selbst aus mehreren Objektdateien/modulen bestehen
- Ziel: Angabe nur eines Names waehrend des Linkens
- Objektdateien muessen zusammengefasst werden

1.9 Bibliotheksarten

Statische Bibliotheken (*.a/*.lib):

- Sammlung von Objektmodulen
- Routinen werden in ausfuehrbaren Code kopiert
- Adressbindung im Linker bei Binaercodegeneration

→ Ergebnis: einzelne Binaerdatei

Dynamische Bibliotheken (*.so/*.dll):

- mehrere Objektmodule (vorgebunden)
- wird vom OS gelanden falls noetig
- Adressbindung erfolgt beim Landen oder waehrnd der Programmausfuehrung

Wird eine dynamische Bibliothek mehrfach angefordert wird sie nur einmal geladen und von mehreren Modulen genutzt

1.10 Loader

Teil des OS (wird bei Programmstart aufgerufen)

Schritt 1 (neuer Ausfuehrungskontext):

- neuer Prozess mit neuem / isolierten Adressraum

Schritt 2 (leadt den Programmcode und Bibliotheken):

- Laden und Kopieren des Programms in den Speicher
- Festlegung von Adressen fuer alle Programmteile

Schritt 3 (Ausfuehren des Linkers):

- Relokation / Anpassung von Adressen
- Verknuepfung von Bibliotheken und Hauptprogramm

Schritt 4 (Sprung in das ausfuehrende Laufzeitssystem)