Einführung in den Compilerbau

Andreas Koch

Wintersemester 2018-2019

Inhaltsverzeichnis

1 Organisatorisches

1.1 Grundlage der Vorlesung

Die Vorlesung basiert fast vollständig auf *Programming Language Processors in Java*[1]. Auszugsweise noch weiteres Material, z. B. zum ANTLR-Parsergenerator.

1.2 Übersichtswerk

Einen guten allgemeinen Überblick, aber im Detail mit anderen Schwerpunkten als diese Vorlesung, bietet *Compilers, 2. Auflage*[2].

1.3 Aufbau der Veranstaltung

Diese Veranstaltung ist logisch in mehrere Teile gegliedert.

Front-End[3] Übersicht, ca. 3 Wochen.

- Lexing und Parsing,
- Zwischendarstellungen.

Middle-End Übersicht, ca. 2 Wochen.

- Semantische- und Kontextanalyse.

Back-End Übersicht, ca. 4 Wochen.

- Laufzeitorganisation,
- Code-Erzeugung.

Front-End-Generatoren Verwendung, ca. 2–3 Wochen.

Java Virtuelle Maschine ca. 1–2 Wochen.

2 Einleitung

2.1 Compiler

Ein Compiler ist eine Schnittstelle zwischen *Mensch* und *Maschine*. Er übersetzt von einer Programmiersprache (Menschenlesbar) in eine Maschinensprache (maschinenlesbar).

[1] von David Watt und Deryck Brown, Prentice-Hall 2000
[2] Von Aho, Sethi, Ullmann, Lam, Addison-Wesley 2006. Auch auf Deutsch verfügbar.
[3] Die ersten drei Teile der Veranstaltung richten sich an die Veranstaltungen *IMT3052* von Ivar Farup, Universität Grøvik, Norwegen; und *Vertalerbouw* von Theu Ruys, Universität Twente, Niederlande.

Programmiersprache Gut für Menschen lesbar. Beispiele für Programmiersprachen sind:

- Smalltalk,
- Java,
- C++.

Maschinensprache Getrimmt auf

- Ausführungsgeschwindigkeit,
- Preis pro Chip, Fläche,
- Energieverbrauch,
- (nur selten) leichte Programmierbarkeit.

2.2 Auswirkung von Compilern

Mit den Eigenschaften entscheidet ein Compiler über die dem Nutzer zugängliche Rechenleistung. Durch gewisse Abwägungen kann ein Compiler ein Programm so kompilieren, dass es am schnellsten läuft oder dass der Resultierende Code an kleinsten ist.

Compiler	Ausführungszeit	Programmgröße
GCC 3.3.6	7,5 ms	13 KB
ICC 9.0	6,5 ms	511 KB

Tabelle 1: Bildkompression auf Dothan CPU, 2 GHz

2.3 Programmiersprachen

Hohe Ebene Smalltalk, Java, C++. Beispiel:

```
let
  var i : Integer;
in
  i := i + 1;
```

Mittlere Ebene Assembly

```
LOAD  R1, (i)
LOADI R2, 1
ADD   R1, R1, R2
STORE R1, (i)
```

Niedrige Ebene Machinensprache

```
01100001 00000110
0111001001000001
1011000100010010
10010001 00000110
```

2.4 Abstraktionsebenen

Auf den unteren Ebenen werden die Beschreibungen immer feiner, da man näher an der Zielmaschine (Hardware) arbeitet.

Der Compiler ist dafür zuständig, Details hinzuzufügen. In den obenstehenden Codebeispielen musste der Compiler unter anderem Register wählen, in denen die Werte währen dem Programmablauf zwischengespeichert werden. Außerdem musste die Adresse der Variable `i`, hinzugefügt werden, wobei hier `00000110` verwendet wurde.

Diese Details werden mithilfe von verschiedensten Algorithmen ergänzt, welche die Programmeigeschaften Analysieren und durch die Synthese von Details die Beschreibung Verfeinern.

3 Zielmaschine

3.1 Auswirkungen der Zielmaschine

Die Zielmaschine hat einen Einfluss auf die Architektur des Compilers. So basiert zum Beispiel die DLX Architektur von John Hennessy und David Patterson auf der MIPS Architektur, und ist nur leicht verändert um diese zu modernisieren. Damit ist es sehr einfach, einen Compiler zu bauen, der für diese Architektur Code generiert.

Etwas komplizierter wird es mit der *TigerSHARC* Architektur von Analog Devices. Dies ist ein Beispiel für ein DSP, also *Digital Signalling Processor*. Es gibt hier zwei *Computational Blocks*, damit parallel Rechnungen ausgeführt werden können. Diese können aber nicht kommunizieren, also muss der Compiler drauf achten, dass auf die Register nur von dem jeweiligen Block aus zugegriffen werden können. Außerdem besitzt diese Architektur separate Rechenblocks, um Adressen zu bestimmen. Ein Compiler für diese Architektur muss also wissen, wie die Architektur aufgebaut ist, um sie effizient zu nutzen.

Am problematischsten wird das aber erst bei extremen Architekturen wie der des IBM/Sony *Cell* Prozessors. Hierbei handelt es sich um eine sehr gewagte, und wie sich leider heraus

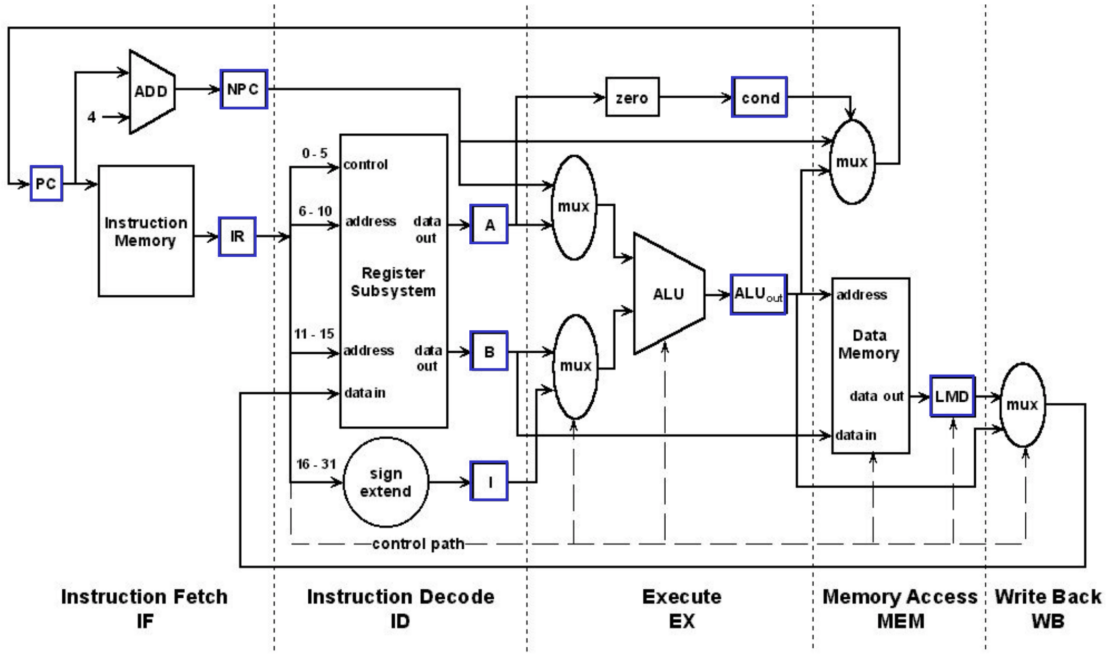

Abbildung 1: Die DLX RISC Prozessorarchitektur

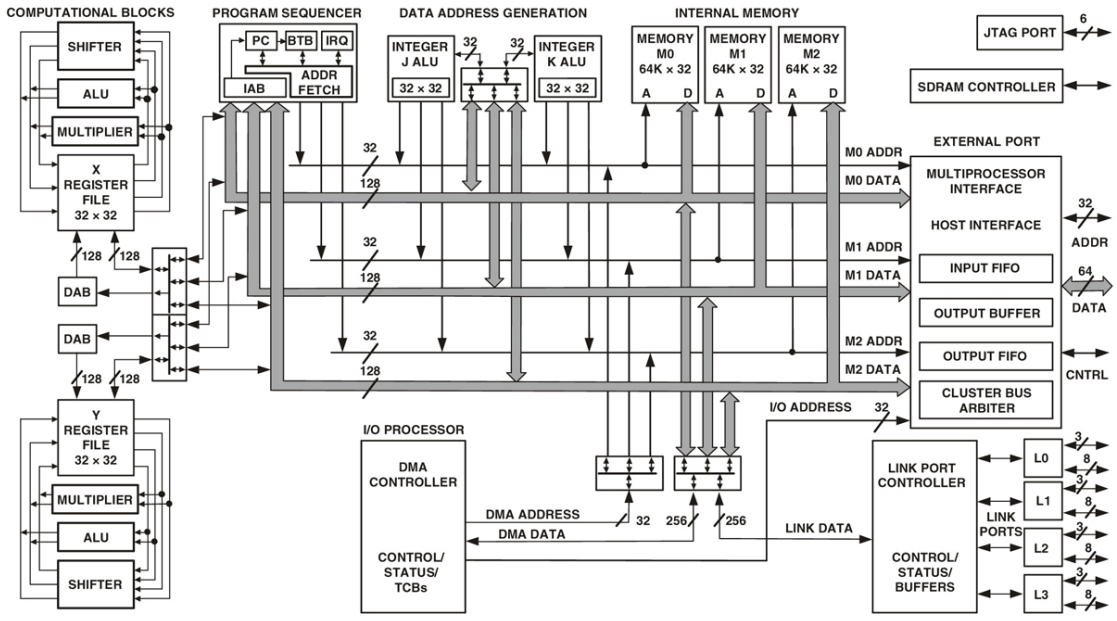

Abbildung 2: Analog Devices TigerSHARC