Compilerkonstruktion

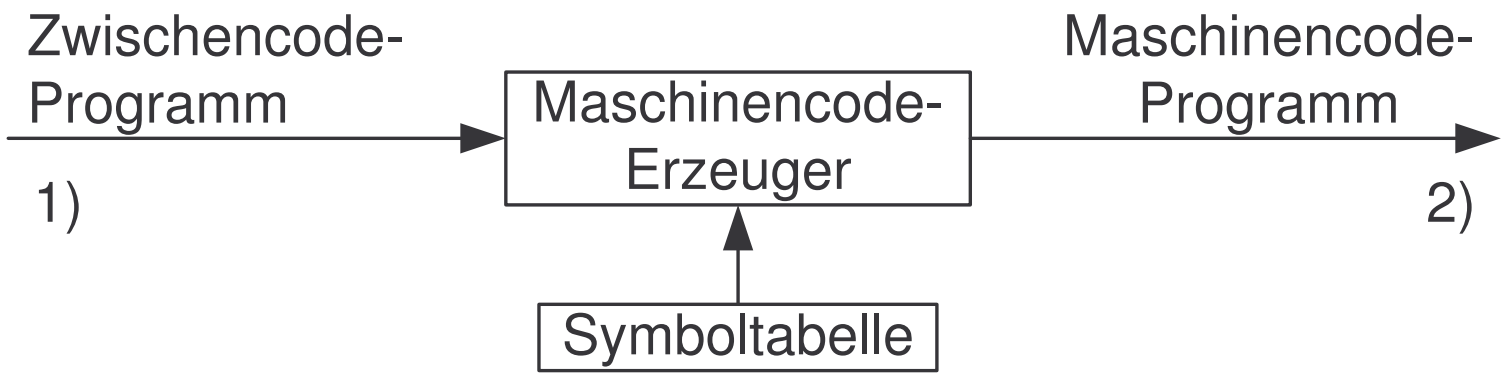
Wintersemester 2015/16

Prof. Dr. R. Parchmann

19. Januar 2016

Maschinencode-Erzeugung

Prinzipieller Aufbau eines Maschinencode-Erzeugers:



Annahmen:

Zu 1) Übersetzung ist soweit durchgeführt, dass

- Werte in der Zwischensprache direkt auf der Zielmaschine dargestellt werden können (integer, real, bits, etc.)
- notwendige Typanpassungen eingefügt wurden.

Zu 2) Verschiedene Formen sind möglich:

- absolutes Maschinenprogramm (direkt ausführbar)
- Objektprogramm (als Eingabe für den Binder)
- Assemblerprogramm

Welche Probleme treten bei der Maschinencode Erzeugung auf?

- Auswahl der Maschinenbefehle speziell bei Zielmaschinen mit einem Befehlscode, der viele Spezialfälle enthält.

Beispiel

Motorola 68000-Prozessoren:

Drei-Adress-Code		Assemblercode	Länge des Befehls
`A := 4`	$\longrightarrow$	`MOVEQ #4,A`	1 Wort
`A := 10`	$\longrightarrow$	`MOVE #10,A`	2 oder 3 Worte
`A := A + 3`	$\longrightarrow$	`ADDQ #3,A`	1 Wort
`A := A + 10`	$\longrightarrow$	`ADDI #10,A`	2 Worte

- Registerzuordnung
  Befehle, deren Operanden in Hardwareregistern vorliegen, sind meist kürzer und schneller abzuarbeiten als äquivalente Befehle, deren Operanden im Speicher abgelegt sind.
  Daher ist es wichtig, häufig benutzte Variablen bzw. Werte in Registern zwischenzuspeichern.
  Erschwerend kommt hinzu, dass manche Befehle nur für spezielle Register oder auch Registerpaare erlaubt sind

Die Zielmaschine

Annahmen:

- Die Hardware ist byte-orientiert,
- Die Maschine besitzt Drei-Adress-Befehle
- Es gibt n allgemeine Register `R0`, `R1`, ..., `Rn`
- Es gibt Lade-Operationen der Form `LD Ri,`*src* mit der Bedeutung, dass der Wert in Adresse *src* in das Zielregister `Ri` geladen wird.
- Es gibt Speicher-Operationen der Form `ST` *dst*`,Ri` mit der Bedeutung, dass der Wert im Register `Ri` in die Speicherzelle *dst* gespeichert wird.

- Es gibt übliche Operationen der Form `OP Ri`, *src1*, *src2*, wobei `OP` eine „übliche“ Operation wie `MOV, ADD, SUB, ...` ist und *src1* und *src2* (nicht notwendigerweise unterschiedliche) Speicherzellen sind. Bei einstelligen Operationen wie etwa `NEG, INC,...` entfällt *src2*.
- Es gibt unbedingte Sprünge der Form `BR` *L*, bei denen der Rechner zum Befehl mit Label *L* verzweigt.
- Es gibt bedingte Sprünge der Form `B`*cond* `Ri`, *L*, dabei steht *cond* für eine der „üblichen“ Bedingungen, die an den Wert im Register `Ri` gestellt werden.

Adressierungsarten des Rechners

- Ein Variablenname bezeichnet den Inhalt der Speicheradresse für diese Variable.
- Ein Konstrukt der Form *num*`(Ri)`, wobei *num* eine Integer-Zahl ist, bezeichnet den Inhalt einer Speicherzelle, deren Adresse man durch Addition der Zahl auf den Inhalt des Registers `Ri` erhält, also *contents*(*num* + *contents*(`Ri`)). Manchmal schreibt man dies auch in der Form *a*`(Ri)`, wobei der Variablenname *a* dann für die Adresse steht, ab der *a* gespeichert ist.

- Ein Konstrukt der Form `*Ri` steht für eine indirekte Adressierung und bezeichnet den Inhalt einer Speicherzelle, deren Adresse in einer Speicherzelle steht, deren Adresse wiederum in Register `Ri` steht. Also bezeichnet `*Ri` den Wert $\mathit{contents}(\mathit{contents}(\mathit{contents}(\texttt{Ri})))$
- Ein Konstrukt der Form $*\mathit{num}\texttt{(Ri)}$ steht für eine indirekte Adressierung und bezeichnet den Inhalt einer Speicherzelle, deren Adresse in einer Speicherzelle steht, deren Adresse sich durch Addition der Zahl *num* auf den Wert im Register `Ri` ergibt. Wir haben also $\mathit{contents}(\mathit{contents}(\mathit{num} + \mathit{contents}(\texttt{Ri})))$

- Ein Konstrukt der Form #*num* beschreibt eine Konstante. Der Befehl `LD R1,#100` lädt zum Beispiel die Konstante 100 in das Register `R1`.

Die Kosten eines Befehls seien definiert als die Länge des Befehls (1 plus der Anzahl der zusätzlichen Speicherzugriffe zum Zugriff auf Konstanten, die nach dem Befehl gespeichert werden) plus der Zahl der Speicherzugriffe auf Operanden der Operation. Die Kosten entsprechen also der Gesamtzahl der Speicherzugriffe.

Beispiel

Befehl	Länge	Kosten
`LD R0,R1`	1	1
`LD R0,M`	2	3
`LD R0,#100`	2	2
`LD R0,100(R1)`	2	3
`LD R0,*R1`	1	2
`LD R0,*100(R1)`	2	4
`ADD R1,*50(R0),*(R2)`	2	6
`BLTZ R1,M`	2	2

Beispiel

Mögliche Übersetzungen von `a := b + c`.
Annahme: Register `R1` und `R2` sind frei:

1. Länge 7, Kosten 10

```
LD  R1,b
LD  R2,c
ADD R1,R1,R2
ST  a,R1
```

2. Länge 5, Kosten 8

```
ADD R1,b,c
ST  a,R1
```

Beispiel

Mögliche Übersetzungen von `a := b + c`.
Annahme: Register `R1` und `R2` sind frei:
Wenn `R3` und `R4` die Adressen von `b` und `c` enthalten, dann:

3. `ADD R1,*R3,*R4` Länge 3, Kosten 6
   `ST a,R1`

Wenn `R1` und `R2` die Werte von `b` und `c` enthalten und der Wert von `b` ab dieser Stelle nicht weiter benötigt wird:

4. `ADD R1,R1,R2` Länge 3, Kosten 4
   `ST a,R1`

Mögliche Übersetzungen anderer Drei-Adress-Befehle:

Annahme: a ist ein Feld von Objekten, die jeweils 8 Bytes belegen.
Die untere Indexgrenze sei 0.
Dann kann man den Befehl `b = a[i]` übersetzen in:

5.
```
LD   R1,i
MUL  R1,R1,#8
LD   R2,a(R1)
ST   b,R2
```
Länge 8, Kosten 11

Einen Befehl der Form `a[i] = b` kann man übersetzen in:

6.
```
LD   R1,b
LD   R2,i
MUL  R2,R2,#8
ST   a(R2),R1
```
Länge 8, Kosten 11

Einen Befehl der Form `if x<y goto L` übersetzt man in:

7. `LD R1,x` — Länge 7, Kosten 9
   `LD R2,y`
   `SUB R1,R1,R2`
   `BLTZ R1,M`

wobei `M` die dem Label `L` zugeordnete Speicheradresse ist. Sind die Variablen `x` und `y` nach diesem Befehl tot, wäre es günstiger, die ersten drei Befehle durch `SUB R1,x,y` zu ersetzen!