Kapitel 6: Befehlsauswahl

Aufgabe

Erzeugen von Assembler/Maschinencode

Themen

- Befehlsauswahl (Baumgrammatik)
- Burg Beispiel

Prozessor-Klassen

CISC

2-Adress-Befehle, mehrere Registerklassen, viele Adressierungsarten

```
<Name>  <Operand1>,<Operand2/Zieloperand>
```

RISC

3-Adress-Befehle, eine Registerklasse, einfache Adressierungsarten
a) Rechenbefehle, alle Operanden in Registern

```
<Name>  <Zieloperand>,<Operand1>,<Operand2>
```

b) Speicherbefehle

```
<Name>  <Register>,<Speicher>
```

Befehlsauswahl

Aufgabe

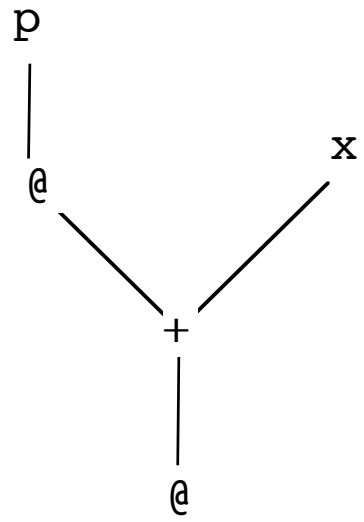
Abbildung von Zwischencode-Befehlen auf Maschinencode-Befehle (wichtig und aufwendig bei CISC-Prozessoren)

Methode

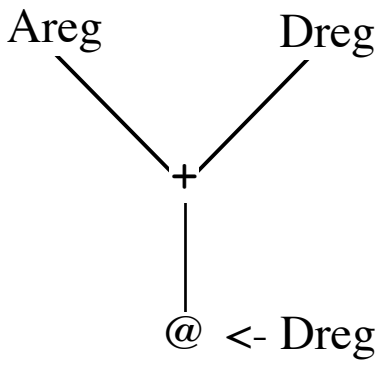
Baumgrammatik

Zwischencode–Baum

C–Ausdruck	Quadrupel	Baum

`(*p)[x]`	`t1 = @ p`
	`t2 = t1 + x`
	`t3 = @ t2`
		

Befehlsbaum

Maschinenbefehl	Baum	Produktion
`move 0(An,Dn),Dn`		Dreg -> @(+(Areg,Dreg))

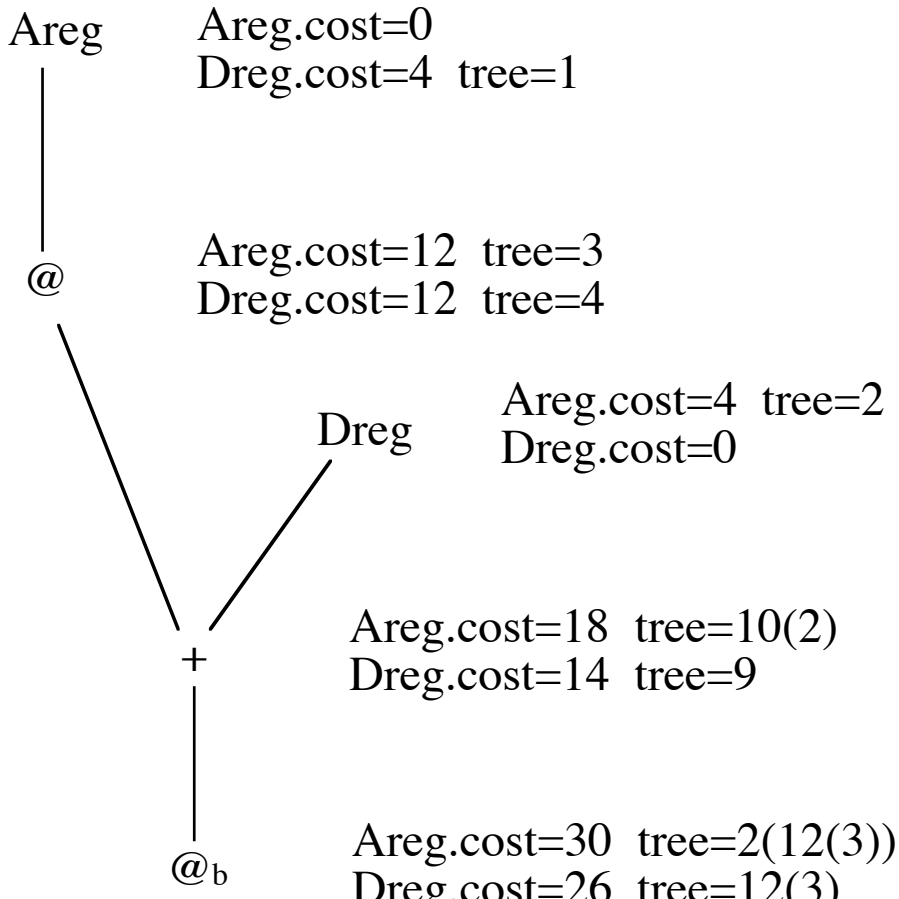
- **Maschinenbefehl als Zwischencode-Teilbaum**
- **Zwischencode-Teilbaum als Grammatik-Produktion**

Befehlsauswahl:
Aus einem gegebenem Zwischencode-Baum jene Teilbäume bestimmen, die einer optimalen Befehlsfolge entsprechen

Baumgrammatik für 68000

Nr	Regel	Befehl	Kosten
1	Dreg -> Areg	`move.l An,Dn`	4
2	Areg -> Dreg	`move.l Dn,An`	4
3	Areg -> @(Areg)	`move.l (An),An`	12
4	Dreg -> @(Areg)	`move.l (An),Dn`	12
5	Dreg -> +(Areg,Dreg)	`add.l An,Dn`	8
6	Dreg -> +(Dreg,Dreg)	`add.l Dn,Dn`	8
7	Areg -> +(Dreg,Areg)	`add.l Dn,An`	8
8	Areg -> +(Areg,Areg)	`add.l An,An`	8
9	Dreg -> +(@(Areg),Dreg)	`add.l (An),Dn`	14
10	Areg -> +(@(Areg),Areg)	`add.l (An),An`	14
11	Dreg -> @$_b$(Areg)	`move.b (An),Dn`	8
12	Dreg -> @$_b$(+(Areg,Dreg))	`move.b 0(An,Dn),Dn`	14
13	Dreg -> @$_b$(+(Areg,Areg))	`move.b 0(An,An),Dn`	14
14	Areg -> an		
15	Dreg -> dn		

Anwendung der Baumgrammatik




```
move.l (An),An
move.b 0(An,Dn),Dn
```

BFE Beispiel

```
%start reg
%term ADD=1 REG=2 NUM=3 ASSIGN=4 ADDASSIGN=5

%%
reg: ASSIGN(reg,reg)    # 1 # printf("movq %s, %s\n",
                              kids[1]->regname, kids[0]->regname);
reg: ASSIGN(reg,num)    # 1 # printf("movq $%d, %s\n",
                              kids[1]->val,kids[0]->regname);
reg: ADDASSIGN(reg,reg) # 1 # printf("addq %s, %s\n",
                              kids[1]->regname,kids[0]->regname);
reg: ADDASSIGN(reg,num) # 1 # printf("addq $%d, %s\n",
                              kids[1]->val,kids[0]->regname);
num: ADD(num,num)       # 0 # bnode->val = kids[0]->val
                                         + kids[1]->val;

reg: REG                # 0
num: NUM                # 0
%%
extern treenode *root;
extern int yyparse(void);
void burm_reduce(NODEPTR_TYPE bnode, int goalnt);
void invoke_burm(NODEPTR_TYPE root) {
        burm_label(root);
        burm_reduce(root, 1);
}
```