

# Kapitel 9: Codeerzeugung

## Aufgabe

Erzeugen von Assembler/Maschinencode

## Themen

- Befehlsauswahl (Baumgrammatik)
- Befehlsanordnung (Abhängigkeitsgraph)
- Registerbelegung (Konfliktgraph)

# 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**

`(*p)[x]`

**Quadrupel**

`t1 = @ p`  
`t2 = t1 + x`  
`t3 = @ t2`

**Baum**

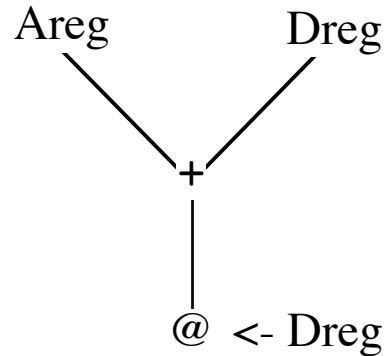


# Befehlsbaum

Maschinenbefehl

move 0 (An, Dn), Dn

Baum



Produktion

Dreg -> @+(Areg,Dreg))

- Maschinenbefehl als Zwischencode–Teilbaum
- Zwischencode–Teilbaum als Grammatik–Produktion

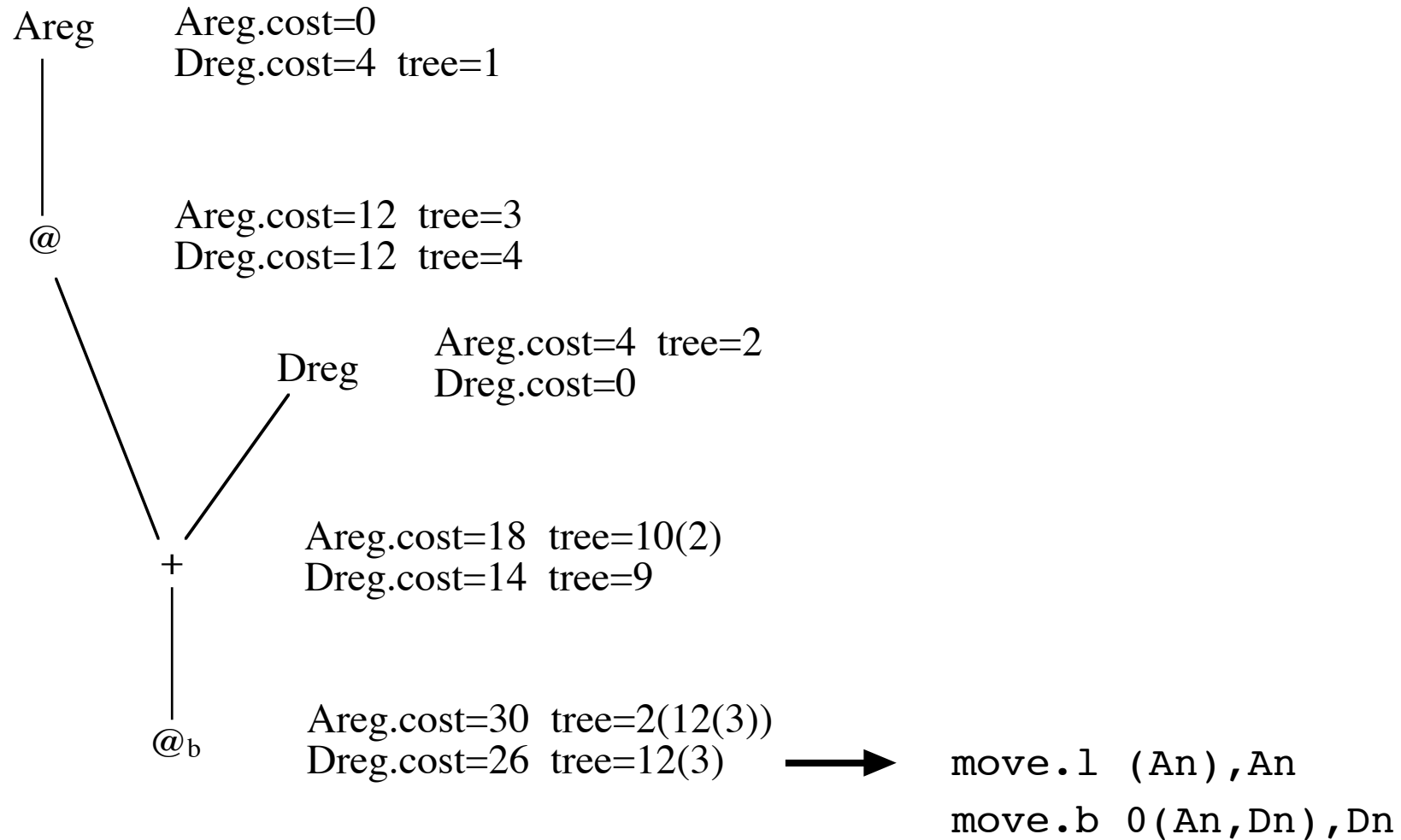
**Befehlsauswahl:**

**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



# Befehlsanordnung

## Aufgabe

Minimierung der Wartezeiten bei RISC-Prozessoren

## Methode

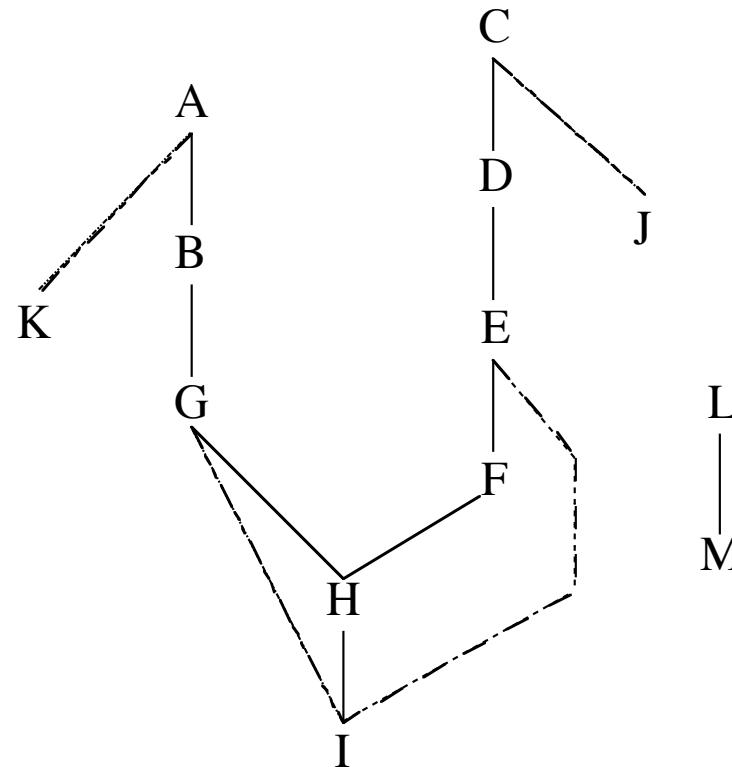
Datenabhängigkeitgraph

# Datenabhängigkeitsgraph

```
for (i=0; i<n; i++) {
    y[iy] = y[iy]+a*x[ix];
    ix += incx;
    iy += incy; }
```

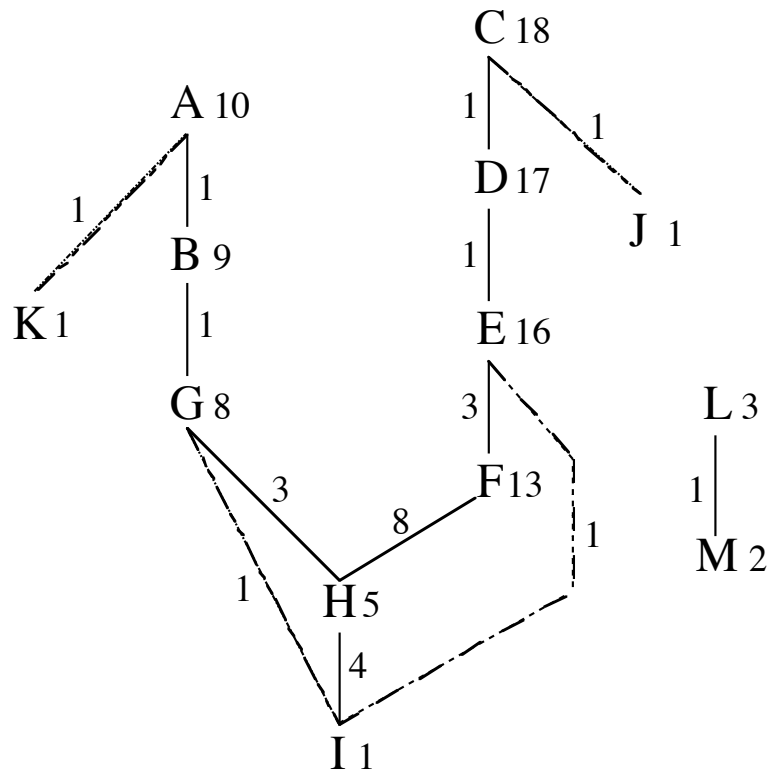
```
$32:
A mul    $14, $4, 8
B addu   $6, $5, $14
C mul    $15, $7, 8
D addu   $24, $15, $8
E l.d    $f4, 0($24)
F mul.d  $f6, $f4, $f12
G l.d    $f8, 0($6)
H add.d  $f10, $f6, $f8
I s.d    $f10, 0($6)
J addu   $7, $7, $9
K addu   $4, $4, $10
L addu   $2, $2, 1
M bne    $11, $2, $32
```

————— Datenfluß  
 - - - - - Lesen-Schreiben  
 - · - · - potentiell



# Anwendung des Abhängigkeitsgraphen

## Graph mit Zyklus-Zahlen



## Umgeordnete Befehlsfolge

	\$32:		
C	mul	\$15, \$7, 8	
D	addu	\$24, \$15, \$8	
E	l.d	\$f4, 0(\$24)	
A	mul	\$14, \$4, 8	
B	addu	\$6, \$5, \$14	
F	mul.d	\$f6, \$f4, \$f12	
G	l.d	\$f8, 0(\$6)	
L	addu	\$2, \$2, 1	
K	addu	\$4, \$4, \$10	
J	addu	\$7, \$7, \$9	
H	add.d	\$f10, \$f6, \$f8	
M	bne	\$11, \$2, \$32	
I	s.d	\$f10, 0(\$6)	

# Registerbelegung

## Aufgabe

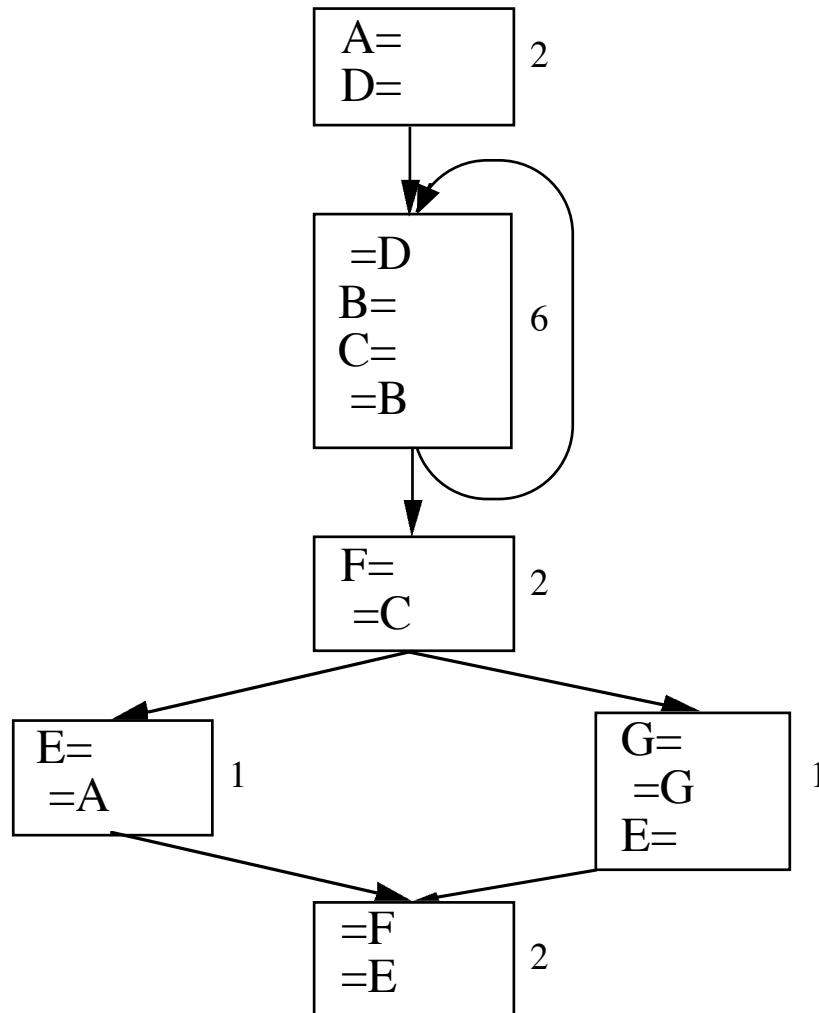
Minimierung der Speicherzugriffe

## Methode

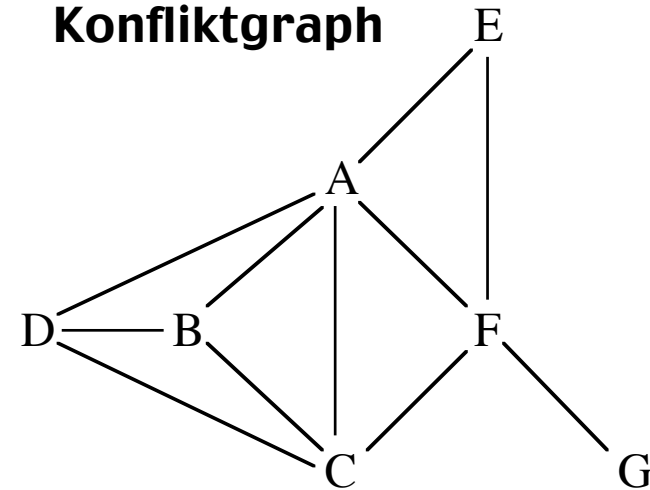
Kontrollfluß-Graph und Konfliktgraph

# Kontrollfluß- und Konfliktgraph

Kontrollflußgraph



Konfliktgraph

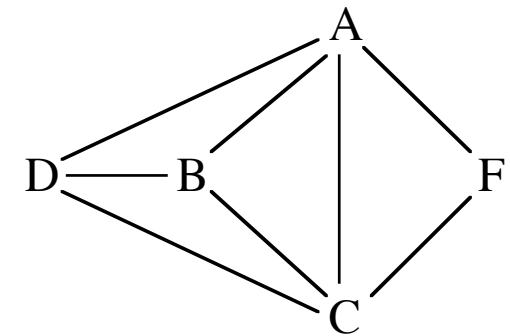
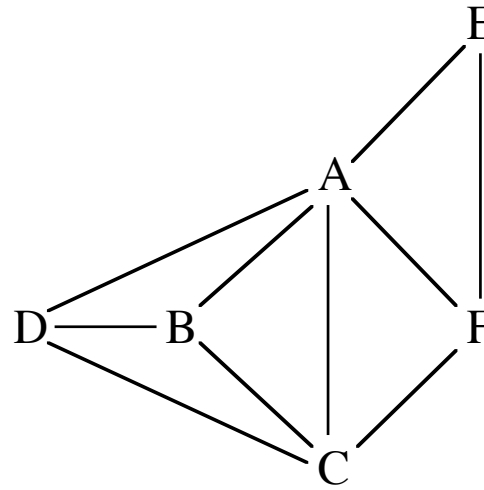
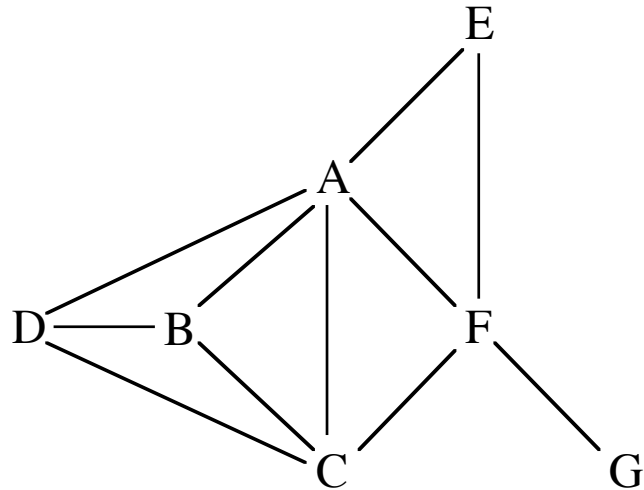


Auslagerungskosten

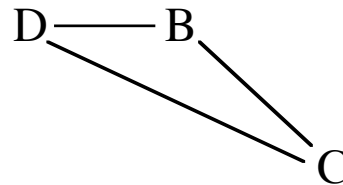
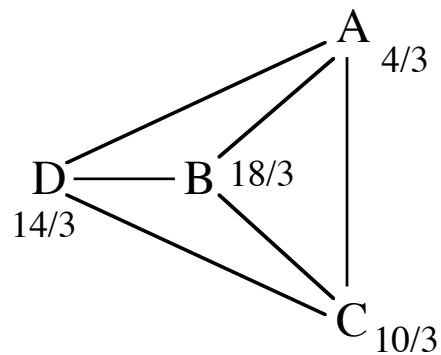
A	4
B	18
C	10
D	14
E	5
F	6
G	3

# Ordnung der Knoten (bei 3 Registern)

a) Knoten mit Grad  $< 3$  entfernen

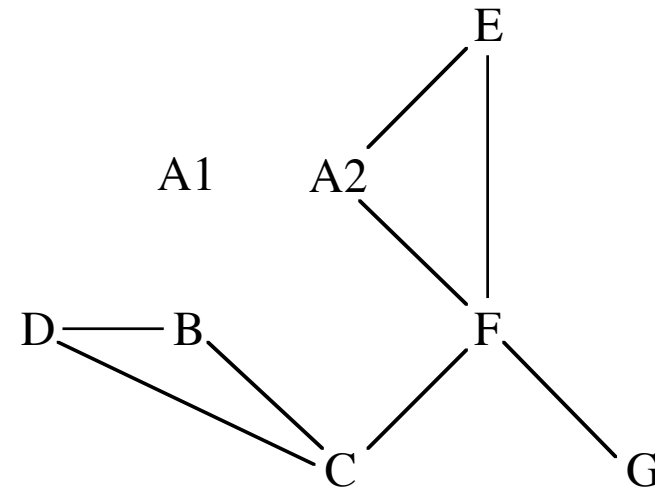
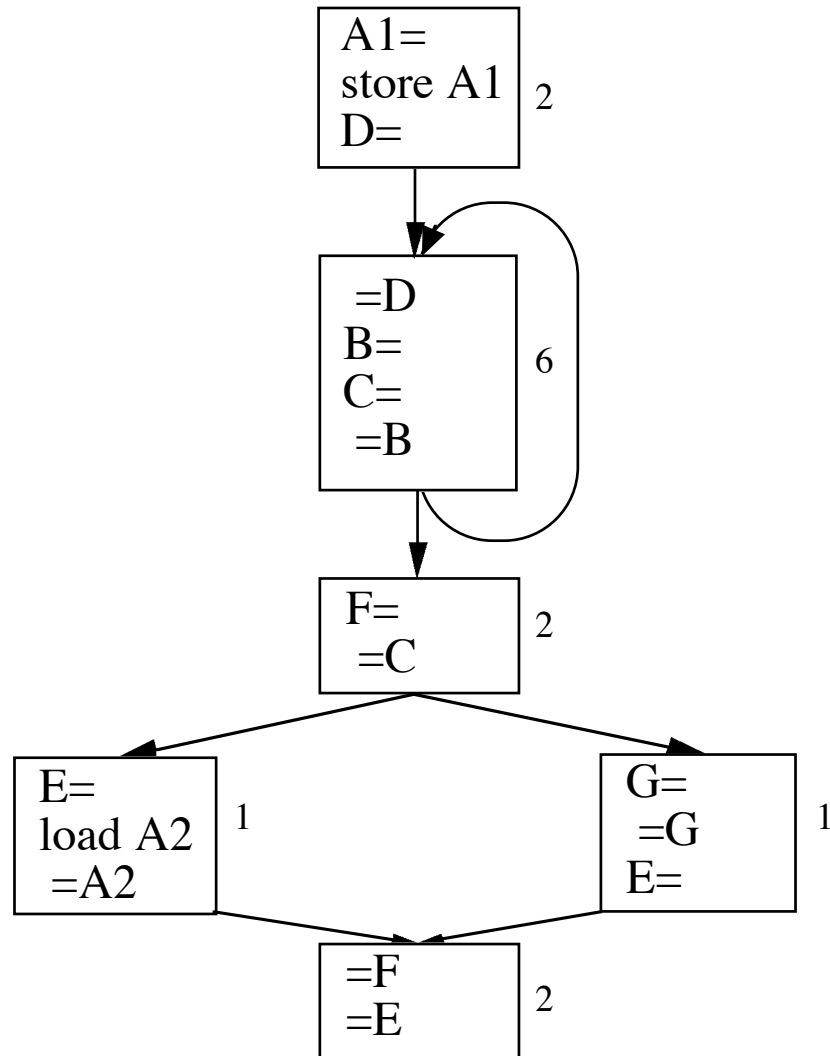


b) Knoten niedrigster Priorität  $k/g$  entfernen



G - E - F - A - D - C - B  
 ←  
**Registerbelegung**

# Auslagerung eines Registers



A1 - G - E - A2 - F - C - B - D



**Registerbelegung**