FH AACHEN
UNIVERSITY OF APPLIED SCIENCES

Codeverbesserung

- **Ziel:**

  compiler-erzeugter Code = handgeschriebener Code

  allgemein für alle Programme der Quellsprache

  speziell für ein Programm

- **Problem:**

  Compiler-erzeugter Code meistens schlechter als handgeschriebener Code

- **aber:**

  Compiler-erzeugter Code meistens automatisiert verbesserbar

- **hier:**

  einige Verfahren zur automatisierten Codeverbesserung

Verbesserungmöglichkeiten

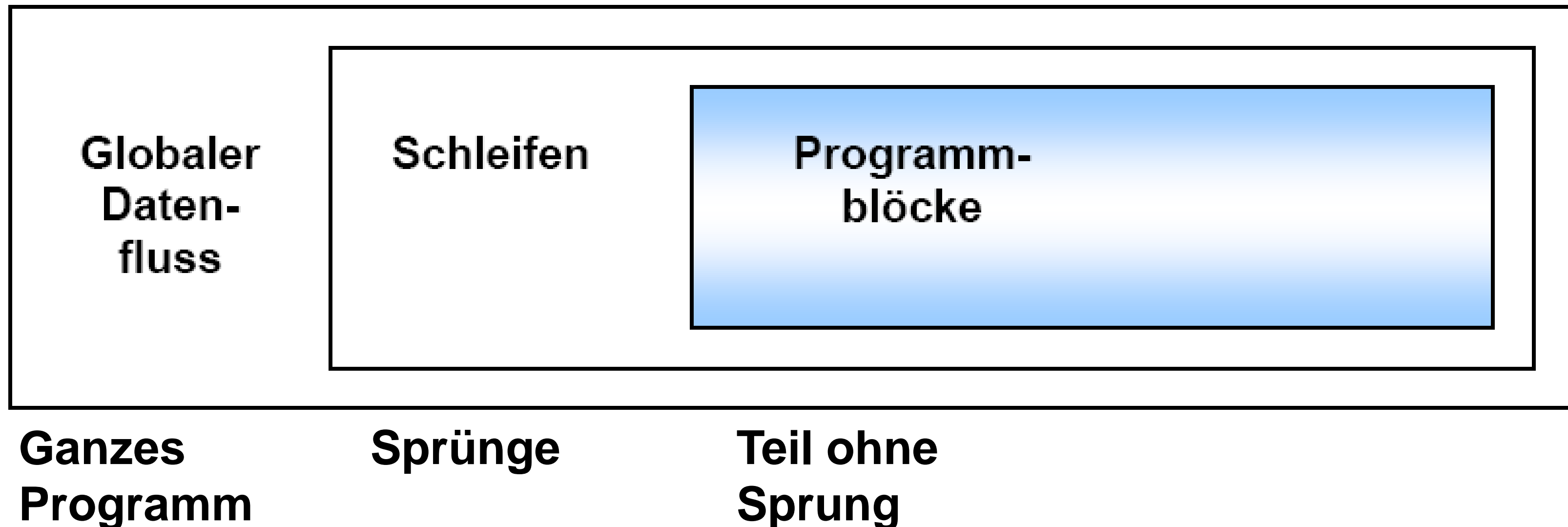
- **Übliche Übersetzungsschritte:**

- **Verbesserungspunkte**
  - Guter Ausgangsalgorithmus (Quellcode)
  - Gute Zwischencodeerzeugung (Frontend)
  - Gute Objektcodeerzeugung (Backend)
  - **Hier: Nachträgliche Verbesserung des erzeugten Code**

FH AACHEN UNIVERSITY OF APPLIED SCIENCES

Verbesserungsstellen

- **Programmaufteilung**



- **Hier: inside out**
  - Programmblöcke
  - Schleifen
  - Datenflussanalyse

Blockverbesserung

- **Ziel: Optimierung des Codes eines Blocks**
- **Angabe von 4 Transformationen, die**
  - Block verkleinern
  - Code verbessern
- **Wichtig: Äquivalenz der erhaltenen Blöcke**
- **Def.: (Block)**

  Ein Block ist ein Code-Abschnitt, der
  - zusammenhängt
  - nicht durch einen Sprung verlassen werden kann
  - in keiner Anweisung Ziel eines Sprunges ist
  - Ausnahmen:
    - Blockanfang und Blockende

FH AACHEN UNIVERSITY OF APPLIED SCIENCES

Bsp.: Blöcke in M32

```
1       ORG 0H
2       MOV @n,6
3       MOV @i,1
4       MOV @fak,1
```

```
        L0
6       MOV TOS,@i
7       MOV TOS,@n
8       MOV R0,TOS
9       MOV R1,TOS
10      CMP R1,R0
11      JGE L0_ENDE
```

```
12      MOV TOS,@fak
13      MOV TOS,@i
14      MOV R1,TOS
15      MOV R0,TOS
16      MUL R0,R1
17      MOV TOS,R0
18      MOV @fak,TOS
19      MOV TOS,@i
20      MOV TOS,1
21      MOV R1,TOS
22      MOV R0,TOS
23      ADD R0,R1
24      MOV TOS,R0
25      MOV @i,TOS
26      MOV TOS,@fak
27      PRN TOS
28      JMP L0
```

```
        L0_ENDE
29      HALT
30      n     DS 1
31      i     DS 1
32      fak   DS 1
33      END
```

Automatische Blockerkennung

- **Blockanfänge sind:**
  - 1. Befehl
  - Jeder Befehl, der Ziel eines Sprunges ist
  - Jeder Befehl nach einem Sprungbefehl
- **Somit automatische Zerlegung des Codes in Blöcke möglich**
- **In M32 Call-Befehl auch Sprungbefehl**
- **Blöcke formal:**

Definition: („Blöcke“)
Variablen: A,B,…
Operationen: f,g,…
Befehle: S $A \leftarrow f(B1,\ldots Br)$

Block: B = <P,I,O> mit P = S1,..,Sn ($n \geq 0$)
mit I ϵ Var (Eingangsvariablen)
mit O ϵ Var (Ausgangsvariablen)

FH AACHEN UNIVERSITY OF APPLIED SCIENCES

Wert & Äquivalenz von Blöcken

FH AACHEN UNIVERSITY OF APPLIED SCIENCES

- **Beispiel-Block:**

  B = < P ; I ; O >
  I = {A,B,C}
  O = {F,G}
  P = (S1) F ← A + A ;
  (S2) G ← F + C ;
  (S3) F ← A + B ;
  (S4) G ← A * B ;

- **Def.: (Wert eines Blocks)**

  Ausgangsvariablen in Abhängigkeit der Eingangsvariablen

- **Bsp.: Wert des Blocks (nur Terme, nichts algebraisches)**

  F = A + B
  G = A * B

- **Äquivalenz** zweier Blöcke

  Gleichheit der Werte der Blöcke A + B != B + A

Scope einer Anweisung (Input-Variable)

FH AACHEN UNIVERSITY OF APPLIED SCIENCES

- **Der Bereich, in dem Anweisung (Input-Var) einen Einfluss hat**
- **Scope der Anweisung Si = A ← f(…) oder von Var A aus I:**
  - Von Si+1
  - Bis zur letzten Anweisung des Blockes mit A auf rechter Seite, so dass A zwischenzeitig nicht neu gesetzt wird
  - Oder bis Blockende O, falls A Ausgangsvariable ist und A nicht zwischenzeitig neu gesetzt wird
- **Scope leer, falls A weder auf rechter Seite noch Ausgangsvar.**
- **Bsp.:**

$B = \langle P, I, O \rangle \quad I = \{B, C\} \quad O = \{E\}$

$P = \{ \quad S1 = A \leftarrow B + C$

$S2 = E \leftarrow B - C$

$S3 = D \leftarrow A * E \}$

Scope(B) = {S1,S2}
Scope(C) = {S1,S2}
Scope(S1) = {S2,S3}
Scope(S2) = {S3,O}
Scope(S3) = { }

4 Blocktransformationen

T1: Entfernung passiven Codes

T2: Elimination redundanter Berechnungen

T3: Umbenennung von Hilfsvariablen

T4: Vertauschung von Anweisungen

T1 & T2 reduzieren Befehlsanzahl

T3 & T4 reduzieren Befehlsanzahl nicht

können trotzdem zu effizienterem Code führen

T1: Entfernung passiven Codes

- **Anweisung mit leerem Scope beeinflusst Block-Wert nicht -> Ihre Entfernung führt zu äquivalentem Block**

Bsp.:

B = < P ; I ; O >

I = {A,B,C}

O = {F,G}

P = S1: F ← A + A ;

S2: G ← F + C ;

S3: F ← A + B ;

S4: G ← A * B ;

Scope(A) = {S1,S2,S3,S4}

Scope(B) = {S1,S2,S3,S4}

Scope(C) = {S1,S2}

Scope(S1) = {S2}

Scope(S2) = {}

Scope(S3) = {S4,O}

Scope(S4) = {O}

S2 kann entfernt werden

Beispiel für T1 (Fortsetzung)

B = < P ; I ; O >

I = {A,B,C}

O = {F,G}

P = S1: F ← A + A ;

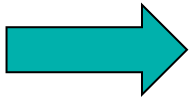
S3: F ← A + B ;

S4: G ← A * B ;

Scope(S1) = {}

Scope(C) = {}

Rest wie vorher ohne S2



S1 und C können entfernt werden

B = < P ; I ; O >

I = {A,B}

O = {F,G}

P = S3: F ← A + B ;

S4: G ← A * B ;

Scope(A) = {S3,S4}

Scope(B) = {S3,S4}

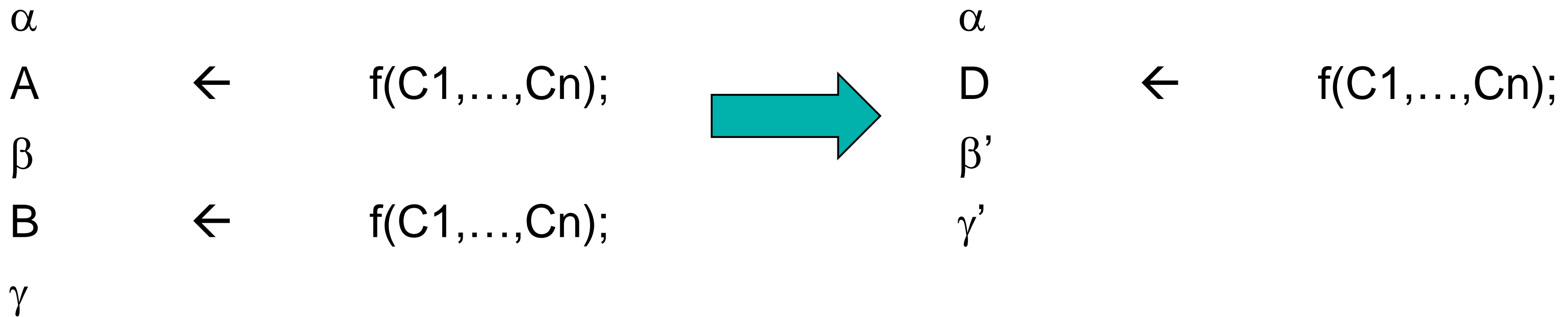
Scope(S3) = {S4,O }

Scope(S4) = {O}

FH AACHEN UNIVERSITY OF APPLIED SCIENCES

T2: Elimination redundanter Berechnungen

- **Fasse Anweisungen mit gleicher rechter Seite zusammen:**



- Mit: D: neue Variable

  β' = β[A/D] im Scope von A ← f(C1,…,Cn);

  γ' = γ[A/D][B/D] im jeweiligen Scope

- **Geht nur, falls folgendes gilt:**
  - A nicht aus {C1, …, Cn}
  - C1, …, Cn in β nicht neu gesetzt

Beispiel für T2

I = {A,B}

O = {F,G}

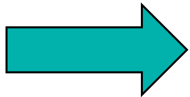
P = S1: S ← A + A ;

S2: F ← A * S ;

S3: R ← F + B ;

S4: T ← A * S ;

S5: G ← T * R ;



S2 und S4 redundant, ersetze F,T durch D, entferne S4

I = {A,B}

O' = {D,G}

P = S1: S ← A + A ;

S2: D ← A * S ;

S3: R ← D + B ;

S5: G ← D * R ;

FH AACHEN UNIVERSITY OF APPLIED SCIENCES

T3: Umbenennung von Hilfsvariablen

- **Trafo**
  - Sei Si: A ← f(C1,…,Cn); eine Anweisung
  - und B eine Variable, die im Scope von Si weder gesetzt noch benutzt wird
  - Dann kann A durch B auf der linken Seite von Si und auf den rechten Seiten des Scopes ersetzt werden

- **keine Verkürzung des Blocks**
- **kann Voraussetzung für andere Trafos sein**
- **kann Registerverwendung verbessern**
  - Bsp.: A ist Speicherplatz und B ist Register

Beispiel für T3

FH AACHEN UNIVERSITY OF APPLIED SCIENCES

I = {A,B}

O = {F}

P = S1: T ← A * B ;

S2: T ← T * A ;

S3: F ← T * T;

Var. S kommt in Scope(S1) = {S2} nicht vor -> Ersetze T durch S

I = {A,B}

O = {F}

P = S1: S ← A * B ;

S2: T ← S * A ;

S3: F ← T * T;

T4: Vertauschung von Anweisungen

- **Trafo**
  - Seien Si: A ← f(B1,…,Bn);
    und Si+1: C ← g(D1,…,Dm);
    zwei aufeinanderfolgende Anweisungen
  - und A nicht in {C,D1,…,Dm}
    und C nicht in {B1,…,Bn}
  - Dann können Si und Si+1 vertauscht werden
- **Wird auch Flipping genannt**
- **keine Verkürzung des Blocks**
- **kann Registerverwendung verbessern**
  - Längeres Halten in Registern erhöht Geschwindigkeit

Beispiel für T4

I = {A,B}
O = {F,G}
P = S1: F ← A + B ;
S2: G ← A * B ;

F und G kommen sonst nicht vor
-> Vertausche S1 & S2

I = {A,B}
O = {F,G}
P = S1: G ← A * B ;
S2: F ← A + B ;

I = {A,B}
O = {F,G}
P = S1: F ← A + B ;
S2: G ← F + B ;

Geht nicht, da F auf rechter Seite von S2

Algebraische Identitäten

- **Bisher: keine algebraischen Regeln angewendet**
  - A + B != B + A
- **Algebraische Regeln:**
  - Kommutativität: a+b = b+a
  - Assoziativität: a+(b+c) = (a+b)+c
  - Distributivität: a*(b+c) = a*b + a*c
  - Idempotenz: -(-a) = a
- **Jetzt Verwendung obiger Regeln**
- **Kann zur Reduktion des Codes führen**
- **Äquivalenz jedoch nicht mehr entscheidbar**

Beispiel für Algebraische Identitäten

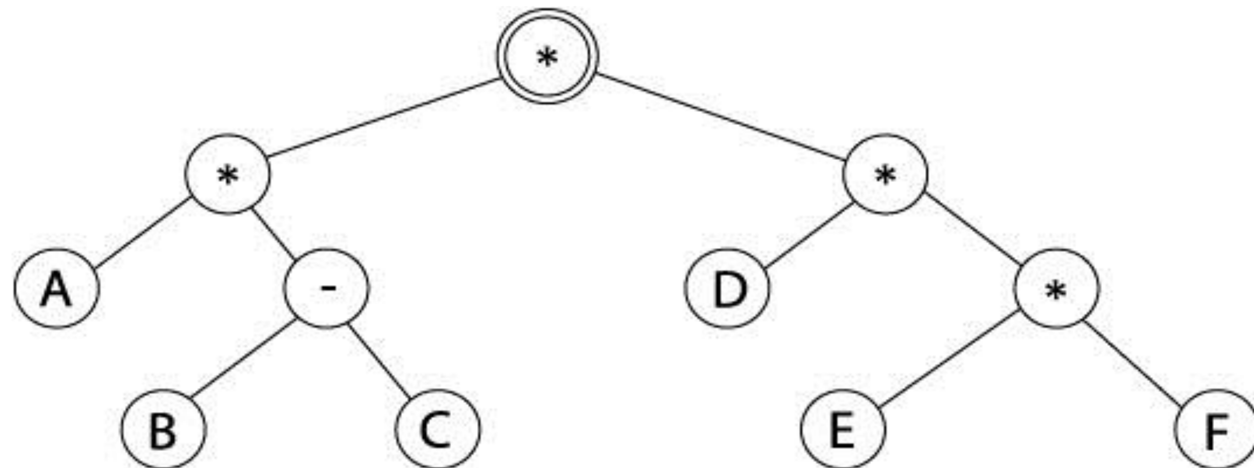
I =	{A,B,C,D,E,F}			
O =	{Y}			
P =	S1:	X1	←	B - C ;
	S2:	X2	←	A * X1;
	S3:	X3	←	E * F;
	S4:	X4	←	D * X3;
	S5:	Y	←	X2 * X4;

Berechneter Ausdruck:

Y = (A*(B-C))*(D*(E*F))

In Akkumaschine:

15 Befehle



FH AACHEN UNIVERSITY OF APPLIED SCIENCES

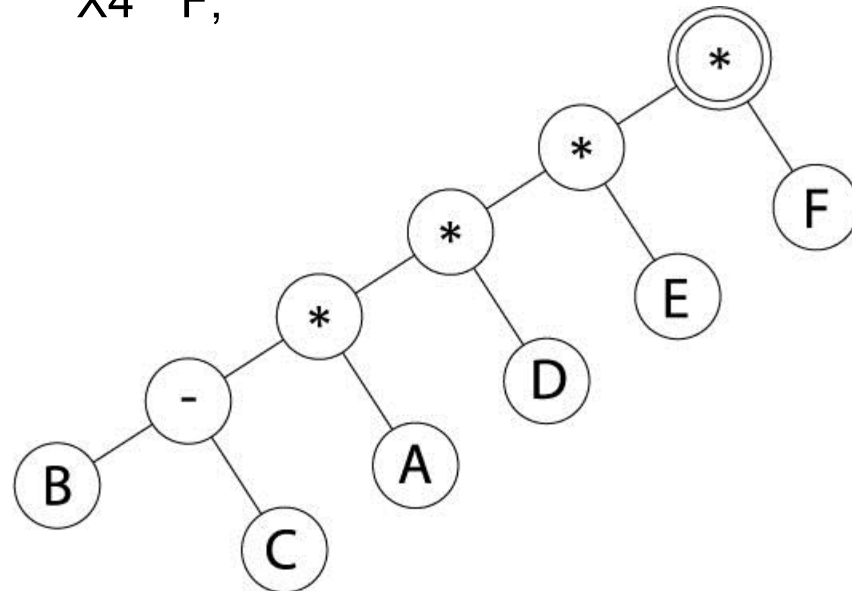
Beispiel für Algebraische Identitäten 2

- **Anwendung algebraischer Identitäten liefert:**

I =	{A,B,C,D,E,F}			
O =	{Y}			
P =	S1:	X1	←	B - C ;
	S2:	X2	←	X1 * A;
	S3:	X3	←	X2 * D;
	S4:	X4	←	X3 * E;
	S5:	Y	←	X4 * F;

Berechneter Ausdruck:
Y = (B-C)*A*D*E*F

In Akkumaschine:
7 Befehle

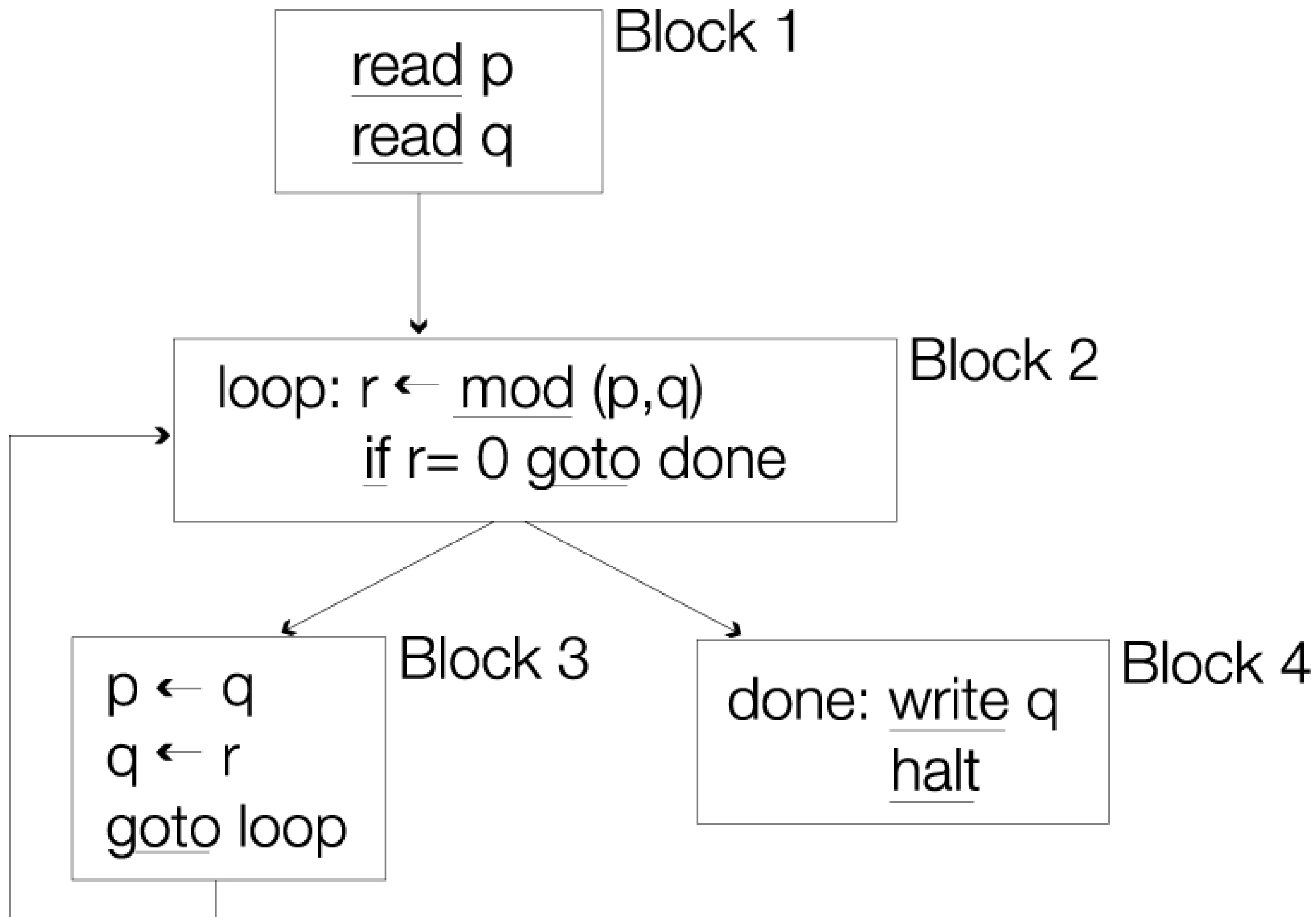


FH AACHEN UNIVERSITY OF APPLIED SCIENCES

Schleifenverbesserung

- **Schleifen in Assembler durch Sprungbefehle implementiert**
- **Werden häufig durchlaufen**
- **Häufig sehr entfernte Sprünge**
- **3 Verbesserungsmöglichkeiten:**
  - Verschieben einer Berechnung aus einer Schleife
  - Induktionsvariablen
  - Abwickeln von Schleifen
- **Im Gegensatz zu Blöcken ohne algebraische Identitäten:**
  - Äquivalenz nicht entscheidbar
  - Keine optimalen Programme
- **Beobachtung:**
  - Wenige Befehle werden sehr oft ausgeführt

Darstellung als Flussgraph

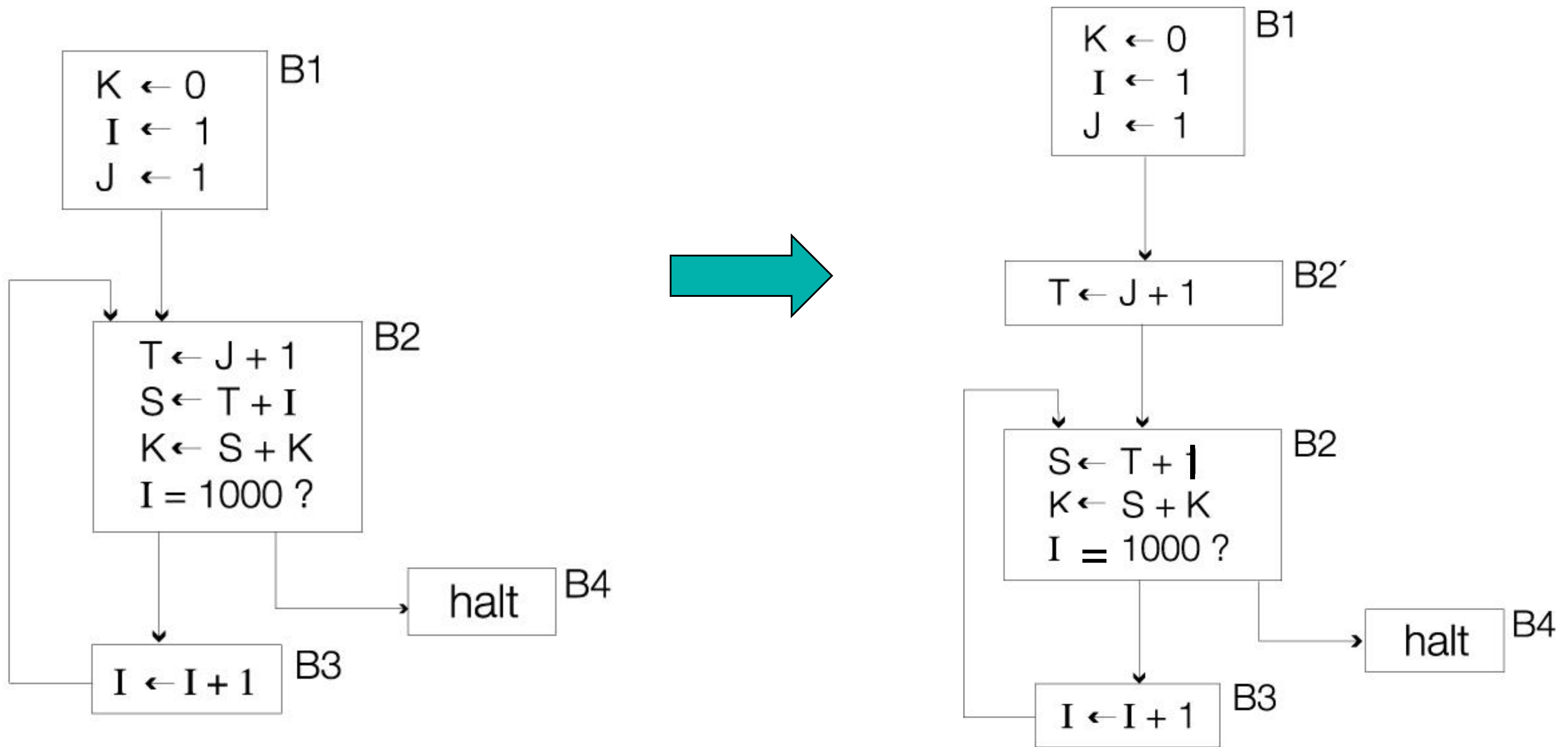


FH AACHEN UNIVERSITY OF APPLIED SCIENCES

Berechnung aus Schleife entfernen

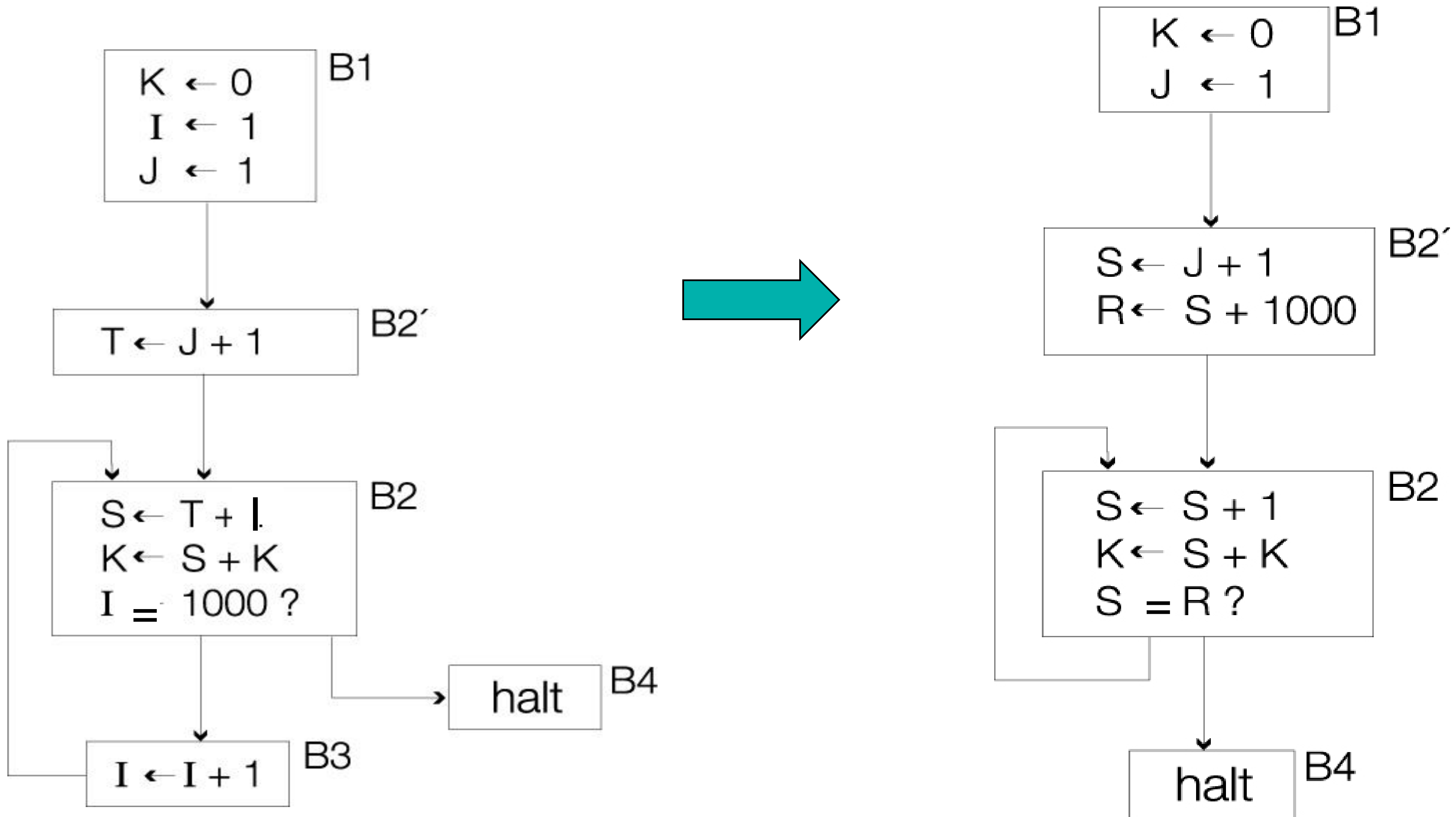
FH AACHEN
UNIVERSITY OF APPLIED SCIENCES

- Invariante Berechnungen aus Schleife rausnehmen



Induktionsvariablen

- Variablen mit linearer Progression
- Schleifenrumpf kürzer -> schneller



FH AACHEN UNIVERSITY OF APPLIED SCIENCES

Schleifen abwickeln

- **Finde Kompromiss zwischen**
  - Verlängerung des Programms und
  - Laufzeit
- **Bsp.:**
  - Verdoppelung des Schleifenrumpfes
  - Halbierung der Anwendung von Sprungbefehlen

Globale Verbesserung

- **Datenflussanalyse, da**

  Trafos zur Schleifenverbesserung Infos über Flussgraphen brauchen
- **Ziemlich komplex, darum hier nur ein Beispiel:**
- **Elimination redundanter Sprungbefehle:**

```
     JMP l1;
     …
l1:  JMP l2;
     …
l2:
```

```
     JMP l2;
     …
l1:  JMP l2;
     …
l2:
```

FH AACHEN UNIVERSITY OF APPLIED SCIENCES

Zusammenfassung

- **Code durch Verbesserung:**
  - verkürzbar
  - schneller
- **Optimaler Code nicht erreichbar**
- **Ggf. von Hand**
  - verbessern
  - programmieren