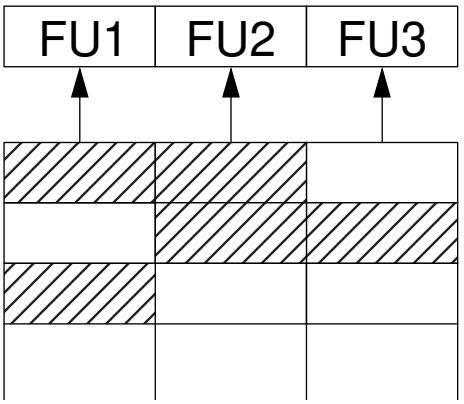
8. Code-Erzeugung für Parallelverarbeitung

Prozessor kann mehrere Instruktionen zugleich ausführen.
Übersetzer ordnet Instruktionen für möglichst kurze Ausführungszeit an (instruction scheduling).

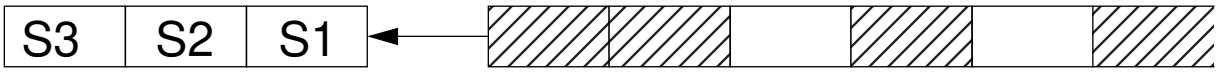
Modelle der Prozessor-Parallelität:

1. Parallele Funktionseinheiten (super scalar, VLIW):



parallel angeordneter Code

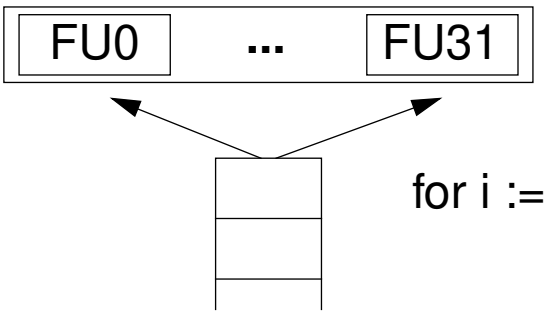
2. Pipeline-Prozessor



sequentiell angeordneter Code

3. Datenparalleler (Vektor-) Prozessor

alle Funktionseinheiten führen zugleich die gleiche Instruktion auf verschiedenen Daten aus



for i := 0 to 31 do c[i] := a[i] + b [i];
eine Instruktion!

Schleifen für parallele Ausführung erkennen und transformieren.

Datenabhängigkeitsgraph

Feingranulare, nutzbare Parallelität von Operationen erkennen.
Sequentielle Anordnung ist überspezifiziert.

Datenabhängigkeitsgraph (für Grundblock):

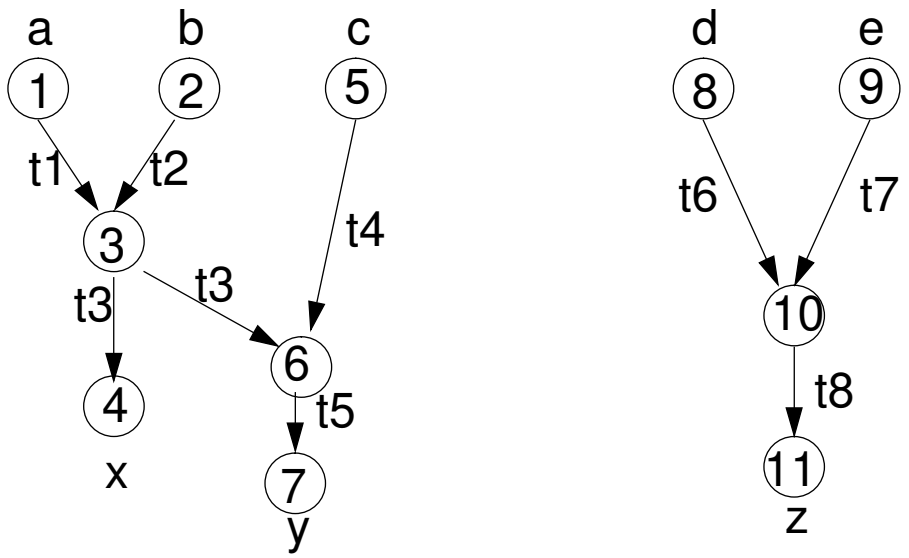
Knoten: Operationen

Kanten: Benutzung von Operationsergebnissen

Beispiel Grundblock:

```
1:    t1  := a
2:    t2  := b
3:    t3  := t1 + t2
4:    x   := t3
5:    t4  := c
6:    t5  := t3 + t4
7:    y   := t5
8:    t6  := d
9:    t7  := e
10:   t8  := t6 + t7
11:   z   := t8
```

Datenabhängigkeitsgraph:



List-Scheduling

gegeben: Datenabhängigkeitsgraph

gesucht: Zuordnung von maximal k Operationen zu jeweils einem Takt, so daß Abhängigkeiten vorwärts weisen (Schedule).

Algorithmus:

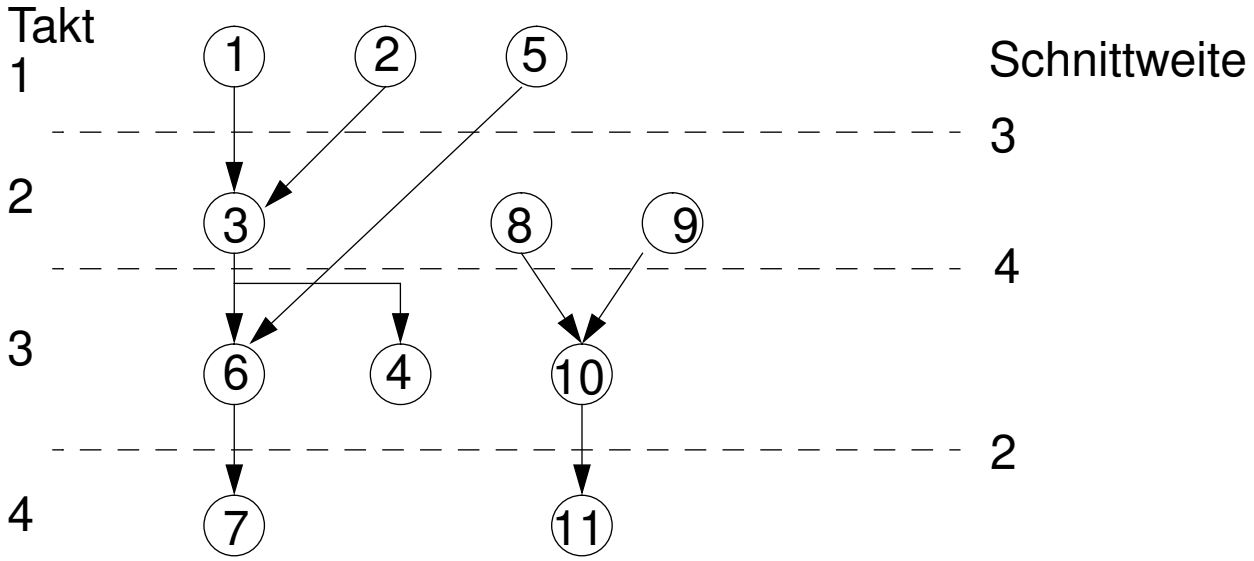
Ready-Liste enthält alle noch nicht angeordneten Operationen, deren Vorgänger alle angeordnet sind.

Iteriere bis alle Operationen angeordnet sind:

Wähle maximal k Operationen für den nächsten Takt aus (Heuristik).

aktualisiere Ready-Liste

Beispiel:



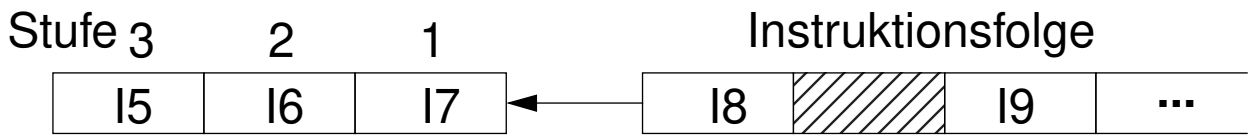
Schnitte im Graphen zeigen **Registerbedarf**.

Bei **unterschiedlichen Ausführungszeiten** von Operationen so auswählen, daß kein Konflikt mit angeordneten Vorgängern.

Algorithmus ist nur **für Bäume optimal**.

Instruktionsanordnung für Pipelining

Instruktions-Pipeline:



Abhängige Instruktionen müssen Mindestabstand haben (hier 2).

Algorithmus: z. B. modifiziertes List-Scheduling

Auswahl der nächsten Instruktion aus der Ready-Liste, so daß

- Mindestabstand zu vorangehenden Instruktionen eingehalten
- Instruktion hat viele Nachfolger (Heuristik)
- Instruktion hat langen Pfad zu Endknoten (Heuristik)

leer-Instruktion einfügen, wenn keine andere wählbar.

Annahme: symbolische Registerzuteilung; reale erfolgt später.

Beispiel:

ohne Umordnung

```
 1:   t1   := a
 2:   t2   := b
      leer
 3:   t3   := t1 + t2
      leer
 4:   x    := t3
 5:   t4   := c
      leer
 6:   t5   := t3 + t4
      leer
 7:   y    := t5
 8:   t6   := d
 9:   t7   := e
      leer
10:   t8   := t6 + t7
      leer
11:   z    := t8
```

mit Umordnung (gem. Alg.)

```
 1:   t1   := a
 2:   t2   := b
 5:   t4   := c
 3:   t3   := t1 + t2
 8:   t6   := d
 9:   t7   := e
 6:   t5   := t3 + t4
10:   t8   := t6 + t7
 4:   x    := t3
 7:   y    := t5
11:   z    := t8
```

Software-Pipelining

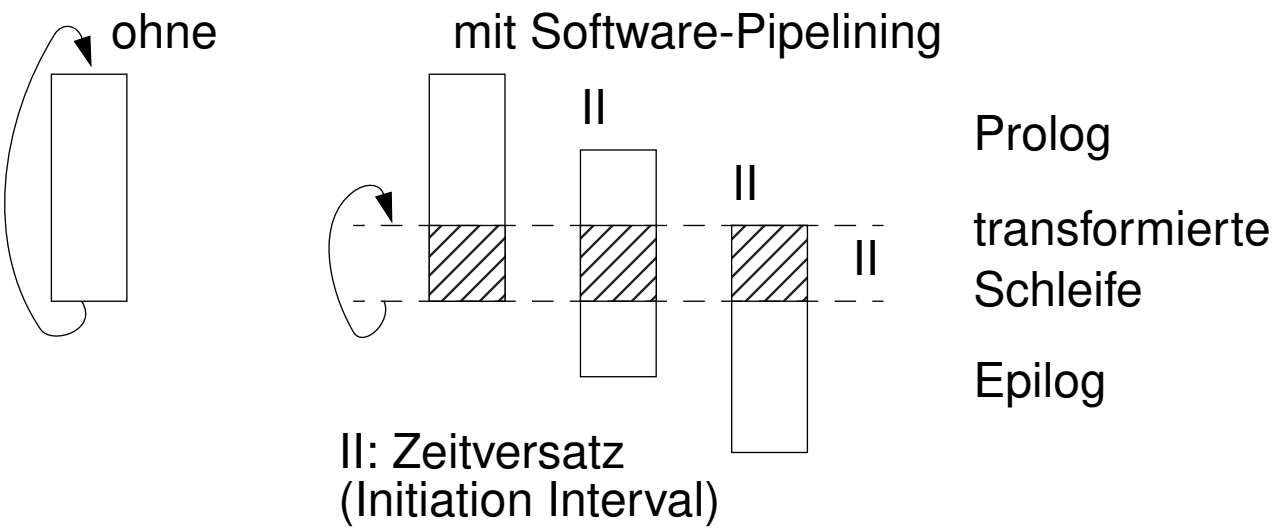
Technik zur **Parallelisierung von Schleifen**

Schleifenrumpf isoliert betrachtet hat wenig Parallelität, daher dünner Schedule.

Prinzip Software-Pipelining:

transformierter Schleifenrumpf bearbeitet mehrere Iterationen parallel, zeitversetzt, daher mehr Parallelität, dichter Schedule

Prolog, Epilog: Anlauf-, Auslauf-Code



Verfahren:

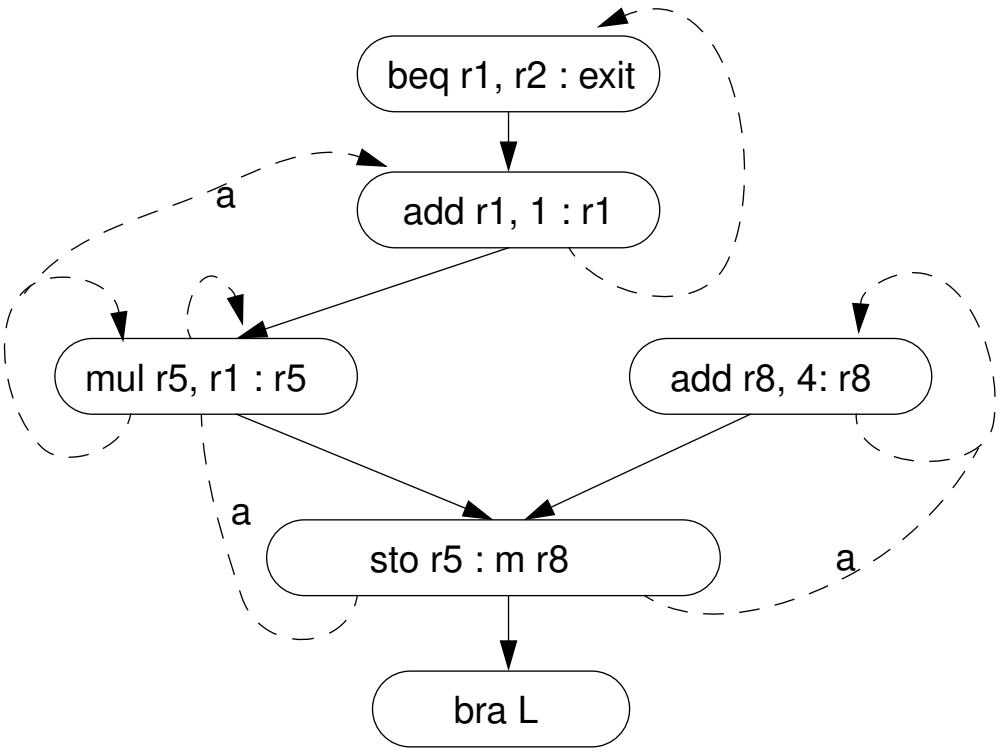
1. Abhängigkeitsgraph für Schleifenrumpf aufstellen; neu: Abhängigkeiten in nachfolgende Iterationen.
2. Schedule herstellen, so daß Iterationen mit möglichst kleinem Zeitversatz (II) überlagert werden können.
3. Neue Schleife, Prolog, Epilog herstellen.

Beispiel Software-Pipelining

Beispiel Fakultätsberechnung:

Programm	seq. Maschinencode
i = 0; f = 1;	
while (i != n)	L: beq r1, r2 : exit
{ i = i + 1;	add r1, 1 : r1
f = f * i;	mul r5, r1 : r5
	add r8, 4 : r8
m[i] = f;	sto r5 : m[r8]
}	bra L

Datenabhängigkeitsgraph:



- - - - → Abhängigkeit in nachfolgende Iteration

u - - -a- - → v Antiabhängigkeit:
u benutzt Wert bevor v ihn überschreibt

Software-Pipelining am Beispiel

Schedule der Schleife:

t	t_m		ADD	MUL	MEM	CTR
0	0	L:				beq r1, r2:exit
1	1		add r1, 1: r1			
2	0		add r8, 4 : r8	mul r5, r1 : r5		
3	1			... mul		
4	0				sto r5 : m r8	
5	1				... sto	
6	0					
7	1					bra L

Software-Pipeline mit II = 2, Prolog, Schleife, Epilog:

t	t_m		ADD	MUL	MEM	CTR
0	0					beq r1;r2:exit
1	1		add r1, 1 : r1			
2	0		add r8, 4 : r8	mul r5, r1 : r5		beq r1; r2 : ex
3	1		add r1, 1 : r1	... mul		
4	0	L:	add r8, 4 : r8	mul r5, r1 : r5	sto r5 : m r8	beq r1; r2 : ex
5	1		add r1, 1 : r1	... mul	... sto	bra L
6	1	ex:		... mul	... sto	
7	0				sto r5 : m r8	
8	1				... sto	
9	0					bra exit