

Rechnertechnik

Prof. Dr. rer. nat. Peter Gerwinski

15. Juni 2021

Rechnertechnik

- 1 Einführung**
- 2 Vom Schaltkreis zum Computer**
- 3 Architekturmerkmale von Prozessoren**
- 4 Der CPU-Stack**
 - 4.1 Implementation
 - 4.2 Unterprogramme
 - 4.3 Register sichern
 - 4.4 Stack-Überläufe
 - 4.5 Puffer-Überläufe
- 5 Anwender-Software**
- 6 Pipelining**
 - 6.1 Konzept
 - 6.2 Arithmetik-Pipelines
 - 6.3 Instruktions-Pipelines

...

5 Anwender-Software

5.6 Puffer-Überläufe

```
#include <stdio.h>
```

```
int main (void)
```

```
{
```

```
    int ID;
```

```
    char buffer[20];
```

```
    printf ("Your_ID, please: ");
```

```
    gets (buffer);
```

```
    sscanf (buffer, "%d", &ID);
```

```
    printf ("Your_name, please: ");
```

```
    gets (buffer);
```

```
    printf ("Hello, %s!\nYour_ID is %d.\n", buffer, ID);
```

```
    return 0;
```

```
}
```

Die Funktion `gets()` prüft nicht, ob `buffer[]` für den eingegebenen Namen ausreicht, und überschreibt ggf. die Variable `ID` sowie die Rücksprungadresse des Funktionsaufrufs von `main()`.




gets() nicht verwenden!

7 Pipelining

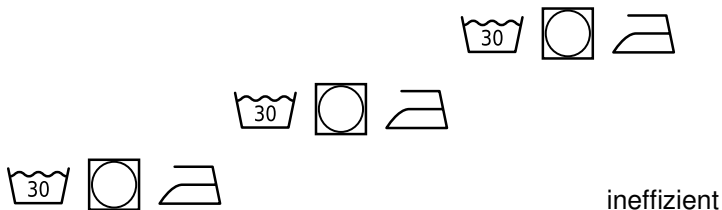
7.1 Konzept

- Aufgabe in Teilaufgaben zerlegen
- Teilaufgaben parallel ausführen

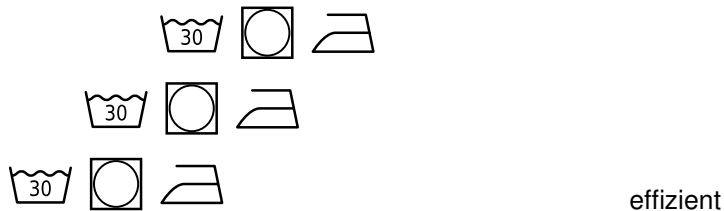
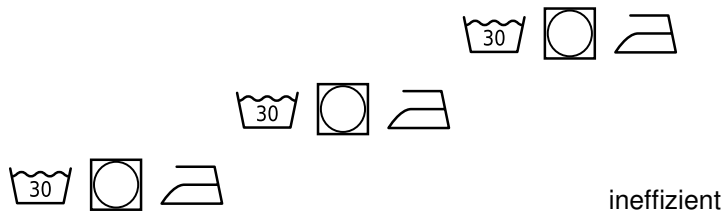
Beispiel: Wäsche waschen

- Teilaufgaben: , , 
- müssen nacheinander ausgeführt werden: Datenfluß
- belegen jeweils 1 Ressource

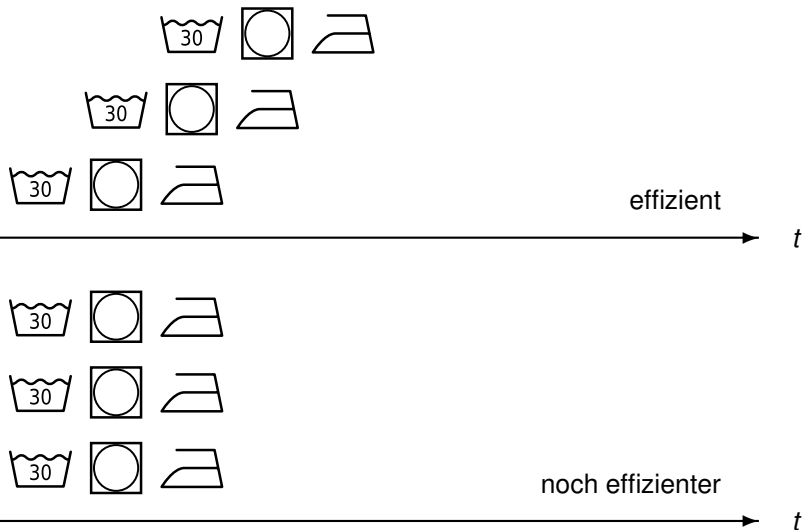
3 Ladungen Wäsche



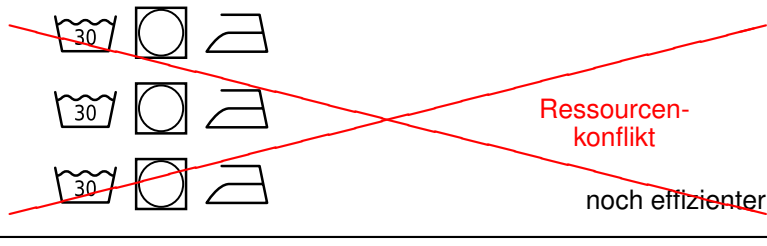
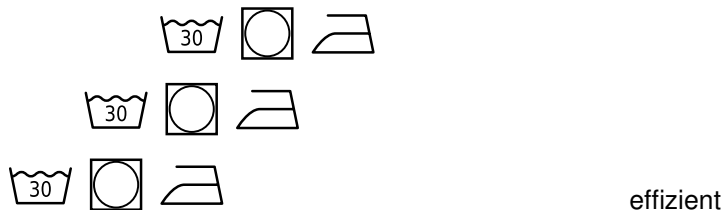
3 Ladungen Wäsche



3 Ladungen Wäsche



3 Ladungen Wäsche



3 Ladungen Wäsche



Ressourcen-
konflikt

noch effizienter

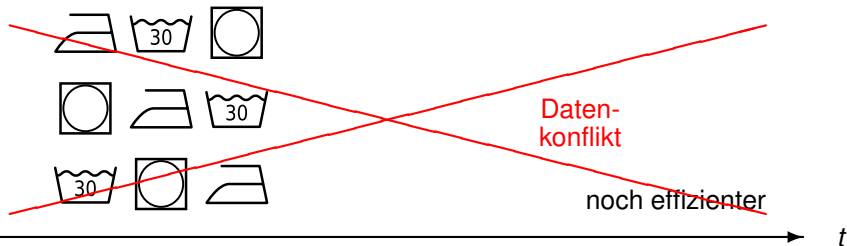
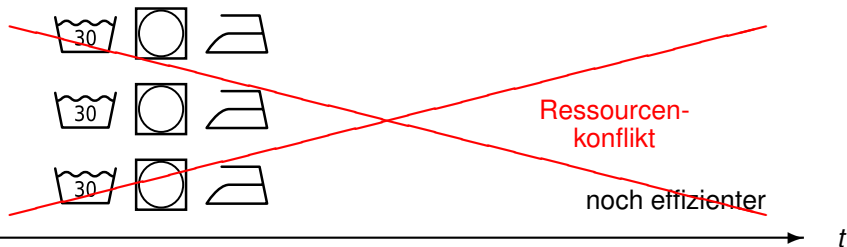
t



noch effizienter

t

3 Ladungen Wäsche



7.2 Arithmetik-Pipelines

„Register-FIFO“

7.2 Arithmetik-Pipelines

„Register-FIFO“

Pseudo-Code: Berechnung von

$$S = \begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix} \cdot \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix} = a_1 \cdot b_1 + a_2 \cdot b_2 + a_3 \cdot b_3$$

mit einer Pipeline der Länge 3

7.2 Arithmetik-Pipelines

„Register-FIFO“

Pseudo-Code: Berechnung von

$$S = \begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix} \cdot \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix} = a_1 \cdot b_1 + a_2 \cdot b_2 + a_3 \cdot b_3$$

mit einer Pipeline der Länge 3

push $a_1 \cdot b_1$

7.2 Arithmetik-Pipelines

„Register-FIFO“

Pseudo-Code: Berechnung von

$$S = \begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix} \cdot \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix} = a_1 \cdot b_1 + a_2 \cdot b_2 + a_3 \cdot b_3$$

mit einer Pipeline der Länge 3

push $a_1 \cdot b_1$



7.2 Arithmetik-Pipelines

„Register-FIFO“

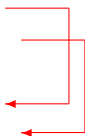
Pseudo-Code: Berechnung von

$$S = \begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix} \cdot \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix} = a_1 \cdot b_1 + a_2 \cdot b_2 + a_3 \cdot b_3$$

mit einer Pipeline der Länge 3

push $a_1 \cdot b_1$

push $a_2 \cdot b_2$



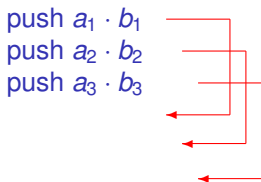
7.2 Arithmetik-Pipelines

„Register-FIFO“

Pseudo-Code: Berechnung von

$$S = \begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix} \cdot \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix} = a_1 \cdot b_1 + a_2 \cdot b_2 + a_3 \cdot b_3$$

mit einer Pipeline der Länge 3



7.2 Arithmetik-Pipelines

„Register-FIFO“

Pseudo-Code: Berechnung von

$$S = \begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix} \cdot \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix} = a_1 \cdot b_1 + a_2 \cdot b_2 + a_3 \cdot b_3$$

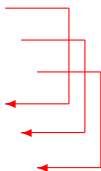
mit einer Pipeline der Länge 3

push $a_1 \cdot b_1$

push $a_2 \cdot b_2$

push $a_3 \cdot b_3$

$s_1 = \text{pop}$



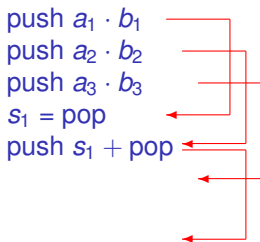
7.2 Arithmetik-Pipelines

„Register-FIFO“

Pseudo-Code: Berechnung von

$$S = \begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix} \cdot \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix} = a_1 \cdot b_1 + a_2 \cdot b_2 + a_3 \cdot b_3$$

mit einer Pipeline der Länge 3



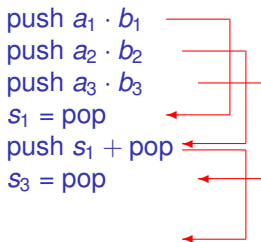
7.2 Arithmetik-Pipelines

„Register-FIFO“

Pseudo-Code: Berechnung von

$$S = \begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix} \cdot \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix} = a_1 \cdot b_1 + a_2 \cdot b_2 + a_3 \cdot b_3$$

mit einer Pipeline der Länge 3



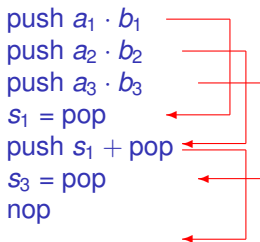
7.2 Arithmetik-Pipelines

„Register-FIFO“

Pseudo-Code: Berechnung von

$$S = \begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix} \cdot \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix} = a_1 \cdot b_1 + a_2 \cdot b_2 + a_3 \cdot b_3$$

mit einer Pipeline der Länge 3



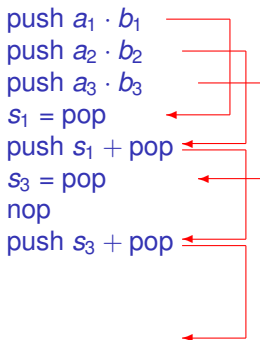
7.2 Arithmetik-Pipelines

„Register-FIFO“

Pseudo-Code: Berechnung von

$$S = \begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix} \cdot \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix} = a_1 \cdot b_1 + a_2 \cdot b_2 + a_3 \cdot b_3$$

mit einer Pipeline der Länge 3



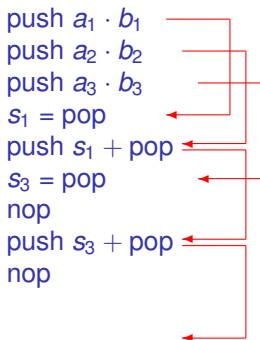
7.2 Arithmetik-Pipelines

„Register-FIFO“

Pseudo-Code: Berechnung von

$$S = \begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix} \cdot \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix} = a_1 \cdot b_1 + a_2 \cdot b_2 + a_3 \cdot b_3$$

mit einer Pipeline der Länge 3



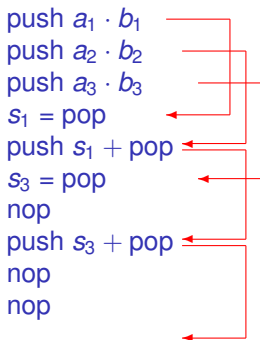
7.2 Arithmetik-Pipelines

„Register-FIFO“

Pseudo-Code: Berechnung von

$$S = \begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix} \cdot \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix} = a_1 \cdot b_1 + a_2 \cdot b_2 + a_3 \cdot b_3$$

mit einer Pipeline der Länge 3



7.2 Arithmetik-Pipelines

„Register-FIFO“

Pseudo-Code: Berechnung von

$$S = \begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix} \cdot \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix} = a_1 \cdot b_1 + a_2 \cdot b_2 + a_3 \cdot b_3$$

mit einer Pipeline der Länge 3

```
graph LR
    subgraph Pipeline
        direction TB
        S1[ ]
        S2[ ]
        S3[ ]
    end
    I1[push a1 · b1] --> S1
    I2[push a2 · b2] --> S1
    I3[push a3 · b3] --> S1
    I4[s1 = pop] --> S1
    I5[push s1 + pop] --> S1
    I6[s3 = pop] --> S1
    I7[nop] --> S1
    I8[push s3 + pop] --> S1
    I9[nop] --> S1
    I10[nop] --> S1
    I11[S = pop] --> S1
    S1 --> S2
    S2 --> S3
    S3 --> S2
    S3 --> S1
```

push $a_1 \cdot b_1$
push $a_2 \cdot b_2$
push $a_3 \cdot b_3$
 $s_1 = \text{pop}$
push $s_1 + \text{pop}$
 $s_3 = \text{pop}$
nop
push $s_3 + \text{pop}$
nop
nop
 $S = \text{pop}$

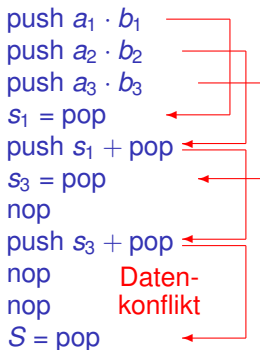
7.2 Arithmetik-Pipelines

„Register-FIFO“

Pseudo-Code: Berechnung von

$$S = \begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix} \cdot \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix} = a_1 \cdot b_1 + a_2 \cdot b_2 + a_3 \cdot b_3$$

mit einer Pipeline der Länge 3



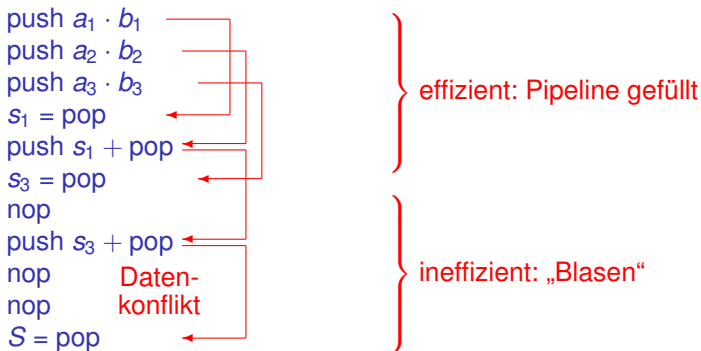
7.2 Arithmetik-Pipelines

„Register-FIFO“

Pseudo-Code: Berechnung von

$$S = \begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix} \cdot \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix} = a_1 \cdot b_1 + a_2 \cdot b_2 + a_3 \cdot b_3$$

mit einer Pipeline der Länge 3



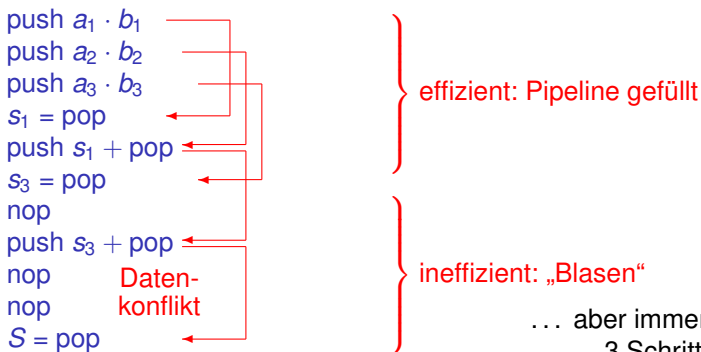
7.2 Arithmetik-Pipelines

„Register-FIFO“

Pseudo-Code: Berechnung von

$$S = \begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix} \cdot \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix} = a_1 \cdot b_1 + a_2 \cdot b_2 + a_3 \cdot b_3$$

mit einer Pipeline der Länge 3



... aber immer noch effizienter als
3 Schritte für jede Operation

Reales Beispiel: Vektor-Addition auf i860

```
.align 8
.globl _vadd
_vadd:
    shr 1,r19,r19
    bte r19,r0,exitadd
    addu 0x000F,r16,r16
    andnot 0x000F,r16,r16
    adds -16,r16,r16
    addu 0x000F,r17,r17
    andnot 0x000F,r17,r17
    adds -16,r17,r17
    addu 0x000F,r18,r18
    andnot 0x000F,r18,r18
    adds -16,r18,r18
    mov -1,r20
```

```
fld.q 16(r16)++,f16
fld.q 16(r17)++,f20
pfadd.dd f16,f20,f0
bla r20,r19,loopadd
pfadd.dd f18,f22,f0
loopadd:
    d.pfadd.dd f0,f0,f0
    fld.q 16(r16)++,f16
    d.pfadd.dd f0,f0,f24
    fld.q 16(r17)++,f20
    d.pfadd.dd f16,f20,f26
    bla r20,r19,loopadd
    d.pfadd.dd f18,f22,f0
    fst.q f24,16(r18)++
    nop
    nop
    nop
exitadd:
    bri r1
    nop
```

Reales Beispiel: Vektor-Addition auf i860


```
.align 8
.globl _vadd
_vadd:
    shr 1,r19,r19
    bte r19,r0,exitadd
    addu 0x000F,r16,r16
    andnot 0x000F,r16,r16
    adds -16,r16,r16
    addu 0x000F,r17,r17
    andnot 0x000F,r17,r17
    adds -16,r17,r17
    addu 0x000F,r18,r18
    andnot 0x000F,r18,r18
    adds -16,r18,r18
    mov -1,r20
```

```
fld.q 16(r16)++,f16
fld.q 16(r17)++,f20
pfadd.dd f16,f20,f0
bla r20,r19,loopadd
pfadd.dd f18,f22,f0
loopadd:
    d.pfadd.dd f0,f0,f0
    fld.q 16(r16)++,f16
    d.pfadd.dd f0,f0,f24
    fld.q 16(r17)++,f20
    d.pfadd.dd f16,f20,f26
    bla r20,r19,loopadd
    d.pfadd.dd f18,f22,f0
    fst.q f24,16(r18)++
    nop
    nop
    nop
exitadd:
    bri r1
    nop
```

Reales Beispiel: Vektor-Addition auf i860

```
.align 8
.globl _vadd
_vadd:
    shr 1,r19,r19
    bte r19,r0,exitadd
    addu 0x000F,r16,r16
    andnot 0x000F,r16,r16
    adds -16,r16,r16
    addu 0x000F,r17,r17
    andnot 0x000F,r17,r17
    adds -16,r17,r17
    addu 0x000F,r18,r18
    andnot 0x000F,r18,r18
    adds -16,r18,r18
    mov -1,r20
```


```
fld.q 16(r16)++,f16
fld.q 16(r17)++,f20
pfadd.dd f16,f20,f0
bla r20,r19,loopadd
pfadd.dd f18,f22,f0
loopadd:
    d.pfadd.dd f0,f0,f0
    fld.q 16(r16)++,f16
    d.pfadd.dd f0,f0,f24
    fld.q 16(r17)++,f20
    d.pfadd.dd f16,f20,f26
    bla r20,r19,loopadd
    d.pfadd.dd f18,f22,f0
    fst.q f24,16(r18)++
    nop
    nop
    nop
exitadd:
    bri r1
    nop
```



Reales Beispiel: Vektor-Addition auf i860

```
.align 8
.globl _vadd
_vadd:
    shr 1,r19,r19
    bte r19,r0,exitadd
    addu 0x000F,r16,r16
    andnot 0x000F,r16,r16
    adds -16,r16,r16
    addu 0x000F,r17,r17
    andnot 0x000F,r17,r17
    adds -16,r17,r17
    addu 0x000F,r18,r18
    andnot 0x000F,r18,r18
    adds -16,r18,r18
    mov -1,r20
```

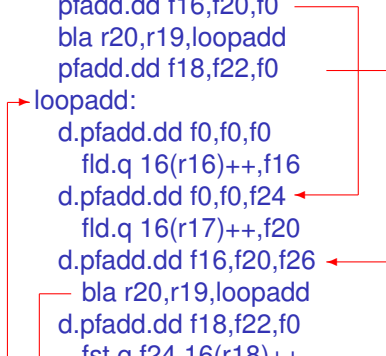
```
fld.q 16(r16)++,f16
fld.q 16(r17)++,f20
pfadd.dd f16,f20,f0
bla r20,r19,loopadd
pfadd.dd f18,f22,f0
loopadd:
    d.pfadd.dd f0,f0,f0
    fld.q 16(r16)++,f16
    d.pfadd.dd f0,f0,f24
    fld.q 16(r17)++,f20
    d.pfadd.dd f16,f20,f26
    bla r20,r19,loopadd
    d.pfadd.dd f18,f22,f0
    fst.q f24,16(r18)++
    nop
    nop
    nop
exitadd:
    bri r1
    nop
```



Reales Beispiel: Vektor-Addition auf i860

```
.align 8
.globl _vadd
_vadd:
    shr 1,r19,r19
    bte r19,r0,exitadd
    addu 0x000F,r16,r16
    andnot 0x000F,r16,r16
    adds -16,r16,r16
    addu 0x000F,r17,r17
    andnot 0x000F,r17,r17
    adds -16,r17,r17
    addu 0x000F,r18,r18
    andnot 0x000F,r18,r18
    adds -16,r18,r18
    mov -1,r20
```

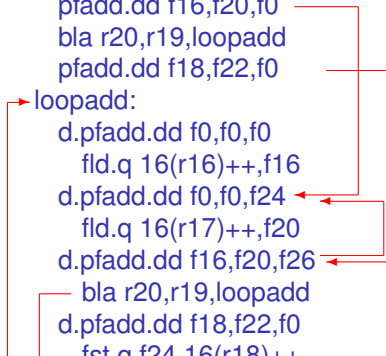
```
fld.q 16(r16)++,f16
fld.q 16(r17)++,f20
pfadd.dd f16,f20,f0
bla r20,r19,loopadd
pfadd.dd f18,f22,f0
loopadd:
    d.pfadd.dd f0,f0,f0
    fld.q 16(r16)++,f16
    d.pfadd.dd f0,f0,f24
    fld.q 16(r17)++,f20
    d.pfadd.dd f16,f20,f26
    bla r20,r19,loopadd
    d.pfadd.dd f18,f22,f0
    fst.q f24,16(r18)++
    nop
    nop
    nop
exitadd:
    bri r1
    nop
```



Reales Beispiel: Vektor-Addition auf i860

```
.align 8
.globl _vadd
_vadd:
    shr 1,r19,r19
    bte r19,r0,exitadd
    addu 0x000F,r16,r16
    andnot 0x000F,r16,r16
    adds -16,r16,r16
    addu 0x000F,r17,r17
    andnot 0x000F,r17,r17
    adds -16,r17,r17
    addu 0x000F,r18,r18
    andnot 0x000F,r18,r18
    adds -16,r18,r18
    mov -1,r20
```

```
fld.q 16(r16)++,f16
fld.q 16(r17)++,f20
pfadd.dd f16,f20,f0
bla r20,r19,loopadd
pfadd.dd f18,f22,f0
loopadd:
    d.pfadd.dd f0,f0,f0
    fld.q 16(r16)++,f16
    d.pfadd.dd f0,f0,f24
    fld.q 16(r17)++,f20
    d.pfadd.dd f16,f20,f26
    bla r20,r19,loopadd
    d.pfadd.dd f18,f22,f0
    fst.q f24,16(r18)++
    nop
    nop
    nop
exitadd:
    bri r1
    nop
```



Reales Beispiel: Vektor-Addition auf i860

```
.align 8
.globl _vadd
_vadd:
    shr 1,r19,r19
    bte r19,r0,exitadd
    addu 0x000F,r16,r16
    andnot 0x000F,r16,r16
    adds -16,r16,r16
    addu 0x000F,r17,r17
    andnot 0x000F,r17,r17
    adds -16,r17,r17
    addu 0x000F,r18,r18
    andnot 0x000F,r18,r18
    adds -16,r18,r18
    mov -1,r20
```

```
fld.q 16(r16)++,f16
fld.q 16(r17)++,f20
pfadd.dd f16,f20,f0
bla r20,r19,loopadd
pfadd.dd f18,f22,f0
loopadd:
    d.pfadd.dd f0,f0,f0
    fld.q 16(r16)++,f16
    d.pfadd.dd f0,f0,f24
    fld.q 16(r17)++,f20
    d.pfadd.dd f16,f20,f26
    bla r20,r19,loopadd
    d.pfadd.dd f18,f22,f0
    fst.q f24,16(r18)++
    nop
    nop
    nop
exitadd:
    bri r1
    nop
```

Reales Beispiel: Vektor-Addition auf i860

```
.align 8
.globl _vadd
_vadd:
    shr 1,r19,r19
    bte r19,r0,exitadd
    addu 0x000F,r16,r16
    andnot 0x000F,r16,r16
    adds -16,r16,r16
    addu 0x000F,r17,r17
    andnot 0x000F,r17,r17
    adds -16,r17,r17
    addu 0x000F,r18,r18
    andnot 0x000F,r18,r18
    adds -16,r18,r18
    mov -1,r20
```

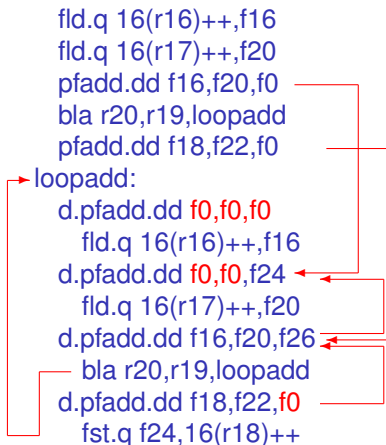
```
fld.q 16(r16)++,f16
fld.q 16(r17)++,f20
pfadd.dd f16,f20,f0
bla r20,r19,loopadd
pfadd.dd f18,f22,f0
loopadd:
    d.pfadd.dd f0,f0,f0
    fld.q 16(r16)++,f16
    d.pfadd.dd f0,f0,f24
    fld.q 16(r17)++,f20
    d.pfadd.dd f16,f20,f26
    bla r20,r19,loopadd
    d.pfadd.dd f18,f22,f0
    fst.q f24,16(r18)++
    nop
    nop
    nop
exitadd:
    bri r1
    nop
```

6mal f0 = 2 Blasen

Reales Beispiel: Vektor-Addition auf i860

```
.align 8
.globl _vadd
_vadd:
    shr 1,r19,r19
    bte r19,r0,exitadd
    addu 0x000F,r16,r16
    andnot 0x000F,r16,r16
    adds -16,r16,r16
    addu 0x000F,r17,r17
    andnot 0x000F,r17,r17
    adds -16,r17,r17
    addu 0x000F,r18,r18
    andnot 0x000F,r18,r18
    adds -16,r18,r18
    mov -1,r20
```

```
fld.q 16(r16)++,f16
fld.q 16(r17)++,f20
pfadd.dd f16,f20,f0
bla r20,r19,loopadd
pfadd.dd f18,f22,f0
loopadd:
    d.pfadd.dd f0,f0,f0
    fld.q 16(r16)++,f16
    d.pfadd.dd f0,f0,f24
    fld.q 16(r17)++,f20
    d.pfadd.dd f16,f20,f26
    bla r20,r19,loopadd
    d.pfadd.dd f18,f22,f0
    fst.q f24,16(r18)++
```



6mal f0 = 2 Blasen

Immerhin: 2 Additionen in 4 Taktzyklen

Reales Beispiel: Vektor-Addition auf i860

```
.align 8
.globl _vadd
_vadd:
    shr 1,r19,r19
    bte r19,r0,exitadd
    addu 0x000F,r16,r16
    andnot 0x000F,r16,r16
    adds -16,r16,r16
    addu 0x000F,r17,r17
    andnot 0x000F,r17,r17
    adds -16,r17,r17
    addu 0x000F,r18,r18
    andnot 0x000F,r18,r18
    adds -16,r18,r18
    mov -1,r20
```

```
fld.q 16(r16)++,f16
fld.q 16(r17)++,f20
pfadd.dd f16,f20,f0
bla r20,r19,loopadd
pfadd.dd f18,f22,f0
loopadd:
    d.pfadd.dd f0,f0,f0
    fld.q 16(r16)++,f16
    d.pfadd.dd f0,f0,f24
    fld.q 16(r17)++,f20
    d.pfadd.dd f16,f20,f26
    bla r20,r19,loopadd
    d.pfadd.dd f18,f22,f0
    fst.q f24,16(r18)++
```

6mal f0 = 2 Blasen

Immerhin: 2 Additionen in 4 Taktzyklen

Dies ist ein *einfaches* Beispiel.

7.3 Instruktions-Pipelines

Ein Prozessor benötigt Zeit, um einen Befehl zu verstehen.

→ Während Befehlsausführung nächste Befehle vorauslesen

.L3:

```
movw r30,r20
add r30,r18
adc r31,r19
mov r24,r18
subi r24,lo8(-(1))
st Z,r24
subi r18,lo8(-(1))
sbci r19,hi8(-(1))
cp r22,r18
cpc r23,r19
brge .L3
ret
```

7.3 Instruktions-Pipelines

Ein Prozessor benötigt Zeit, um einen Befehl zu verstehen.

→ Während Befehlsausführung nächste Befehle vorauslesen

.L3:

movw r30,r20

add r30,r18

adc r31,r19

mov r24,r18

subi r24,lo8(-(1))

st Z,r24

subi r18,lo8(-(1))

sbc r19,hi8(-(1))

cp r22,r18

cpc r23,r19

brge .L3

ret

7.3 Instruktions-Pipelines

Ein Prozessor benötigt Zeit, um einen Befehl zu verstehen.

→ Während Befehlsausführung nächste Befehle vorauslesen

.L3:

movw r30,r20

add r30,r18

adc r31,r19

mov r24,r18

subi r24,lo8(-(1))

st Z,r24

subi r18,lo8(-(1))

sbc r19,hi8(-(1))

cp r22,r18

cpc r23,r19

brge .L3

ret

7.3 Instruktions-Pipelines

Ein Prozessor benötigt Zeit, um einen Befehl zu verstehen.

→ Während Befehlsausführung nächste Befehle vorauslesen

.L3:

movw r30,r20

add r30,r18

adc r31,r19

mov r24,r18

subi r24,lo8(-(1))

st Z,r24

subi r18,lo8(-(1))

sbc r19,hi8(-(1))

cp r22,r18

cpc r23,r19

brge .L3

ret

7.3 Instruktions-Pipelines

Ein Prozessor benötigt Zeit, um einen Befehl zu verstehen.

→ Während Befehlsausführung nächste Befehle vorauslesen

.L3:

```
movw r30,r20
add r30,r18
adc r31,r19
mov r24,r18
subi r24,lo8(-(1))
st Z,r24
subi r18,lo8(-(1))
sbci r19,hi8(-(1))
cp r22,r18
cpc r23,r19
brge .L3
ret
```

7.3 Instruktions-Pipelines

Ein Prozessor benötigt Zeit, um einen Befehl zu verstehen.

→ Während Befehlsausführung nächste Befehle vorauslesen

.L3:

```
movw r30,r20
add r30,r18
adc r31,r19
mov r24,r18
subi r24,lo8(-(1))
st Z,r24
subi r18,lo8(-(1))
sbci r19,hi8(-(1))
cp r22,r18
cpc r23,r19
brge .L3
ret
```

7.3 Instruktions-Pipelines

Ein Prozessor benötigt Zeit, um einen Befehl zu verstehen.

→ Während Befehlsausführung nächste Befehle vorauslesen

.L3:


```
movw r30,r20
add r30,r18
adc r31,r19
mov r24,r18
subi r24,lo8(-(1))
st Z,r24
subi r18,lo8(-(1))
sbci r19,hi8(-(1))
cp r22,r18
cpc r23,r19
brge .L3
ret
```

7.3 Instruktions-Pipelines

Ein Prozessor benötigt Zeit, um einen Befehl zu verstehen.

→ Während Befehlsausführung nächste Befehle vorauslesen

.L3:



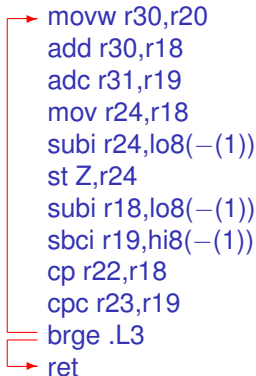
```
movw r30,r20
add r30,r18
adc r31,r19
mov r24,r18
subi r24,lo8(-(1))
st Z,r24
subi r18,lo8(-(1))
sbci r19,hi8(-(1))
cp r22,r18
cpc r23,r19
brge .L3
ret
```

7.3 Instruktions-Pipelines

Ein Prozessor benötigt Zeit, um einen Befehl zu verstehen.

→ Während Befehlsausführung nächste Befehle vorauslesen

.L3:



```
→ movw r30,r20
  add r30,r18
  adc r31,r19
  mov r24,r18
  subi r24,lo8(-(1))
  st Z,r24
  subi r18,lo8(-(1))
  sbci r19,hi8(-(1))
  cp r22,r18
  cpc r23,r19
  brge .L3
→ ret
```

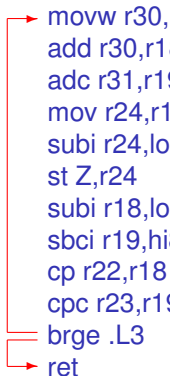
The diagram illustrates a loop structure in assembly code. A red arrow points from the 'ret' instruction back to the 'movw r30,r20' instruction, indicating a branch back to the start of the loop body.

7.3 Instruktions-Pipelines

Ein Prozessor benötigt Zeit, um einen Befehl zu verstehen.

→ Während Befehlsausführung nächste Befehle vorauslesen

.L3:



```
→ movw r30,r20
  add r30,r18
  adc r31,r19
  mov r24,r18
  subi r24,lo8(-(1))
  st Z,r24
  subi r18,lo8(-(1))
  sbci r19,hi8(-(1))
  cp r22,r18
  cpc r23,r19
  brge .L3
→ ret
```

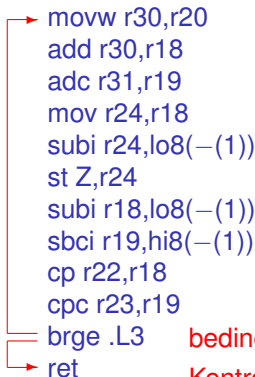
bedingter Sprung: Welche Befehle vorauslesen?

7.3 Instruktions-Pipelines

Ein Prozessor benötigt Zeit, um einen Befehl zu verstehen.

→ Während Befehlsausführung nächste Befehle vorauslesen

.L3:



```
→ movw r30,r20
  add r30,r18
  adc r31,r19
  mov r24,r18
  subi r24,lo8(-(1))
  st Z,r24
  subi r18,lo8(-(1))
  sbci r19,hi8(-(1))
  cp r22,r18
  cpc r23,r19
  brge .L3
→ ret
```

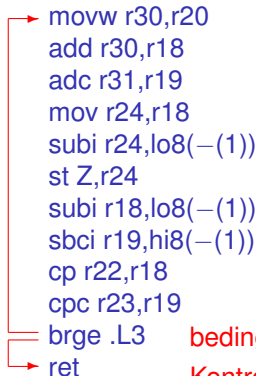
bedingter Sprung: Welche Befehle vorauslesen?
Kontrollflußkonflikt

7.3 Instruktions-Pipelines

Ein Prozessor benötigt Zeit, um einen Befehl zu verstehen.

→ Während Befehlsausführung nächste Befehle vorauslesen

.L3:



```
→ movw r30,r20
  add r30,r18
  adc r31,r19
  mov r24,r18
  subi r24,lo8(-(1))
  st Z,r24
  subi r18,lo8(-(1))
  sbci r19,hi8(-(1))
  cp r22,r18
  cpc r23,r19
  brge .L3
→ ret
```

bedingter Sprung: Welche Befehle vorauslesen?
Kontrollflußkonflikt

Lösungsansatz: Zweigvorhersage

7.3 Instruktions-Pipelines

Zweigvorhersage – Branch Prediction

7.3 Instruktions-Pipelines

Zweigvorhersage – Branch Prediction

- Sprünge nach oben sind Schleifen: „Ja“
Sprünge nach unten sind Auswahl-Verzweigungen: „Nein“

7.3 Instruktions-Pipelines

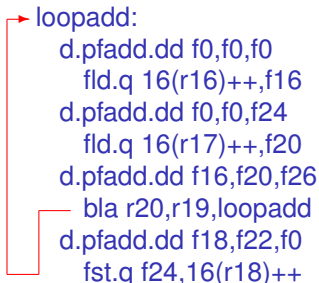
Zweigvorhersage – Branch Prediction

- Sprünge nach oben sind Schleifen: „Ja“
Sprünge nach unten sind Auswahl-Verzweigungen: „Nein“
- Delayed Branches: Sprungbefehl verspätet ausführen
→ Optimierung manuell oder durch Compiler

7.3 Instruktions-Pipelines

Zweigvorhersage – Branch Prediction

- Sprünge nach oben sind Schleifen: „Ja“
Sprünge nach unten sind Auswahl-Verzweigungen: „Nein“
- Delayed Branches: Sprungbefehl verspätet ausführen
→ Optimierung manuell oder durch Compiler

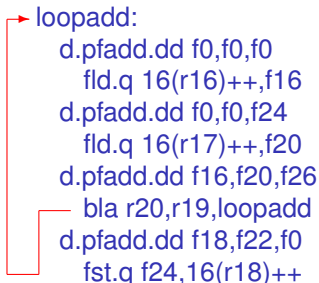


```
loopadd:
    d.pfadd.dd f0,f0,f0
    fld.q 16(r16)++,f16
    d.pfadd.dd f0,f0,f24
    fld.q 16(r17)++,f20
    d.pfadd.dd f16,f20,f26
    bla r20,r19,loopadd
    d.pfadd.dd f18,f22,f0
    fst.q f24,16(r18)++
```

7.3 Instruktions-Pipelines

Zweigvorhersage – Branch Prediction

- Sprünge nach oben sind Schleifen: „Ja“
Sprünge nach unten sind Auswahl-Verzweigungen: „Nein“
- Delayed Branches: Sprungbefehl verspätet ausführen
→ Optimierung manuell oder durch Compiler



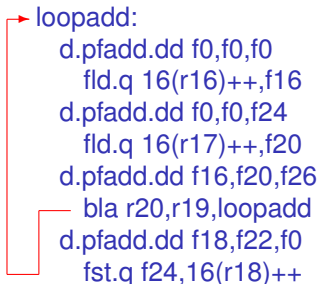
```
loopadd:
    d.pfadd.dd f0,f0,f0
    fld.q 16(r16)++,f16
    d.pfadd.dd f0,f0,f24
    fld.q 16(r17)++,f20
    d.pfadd.dd f16,f20,f26
    bla r20,r19,loopadd
    d.pfadd.dd f18,f22,f0
    fst.q f24,16(r18)++
```

- Branch History Table: Sprünge merken

7.3 Instruktions-Pipelines

Zweigvorhersage – Branch Prediction

- Sprünge nach oben sind Schleifen: „Ja“
Sprünge nach unten sind Auswahl-Verzweigungen: „Nein“
- Delayed Branches: Sprungbefehl verspätet ausführen
→ Optimierung manuell oder durch Compiler



```
→ loopadd:
    d.pfadd.dd f0,f0,f0
    fld.q 16(r16)++,f16
    d.pfadd.dd f0,f0,f24
    fld.q 16(r17)++,f20
    d.pfadd.dd f16,f20,f26
    bla r20,r19,loopadd
    d.pfadd.dd f18,f22,f0
    fst.q f24,16(r18)++
```

- Branch History Table: Sprünge merken
- ...

7 Pipelining

Zusammenfassung

- Teilaufgaben parallel ausführen
- Arithmetik-Pipelines führen Berechnungen parallel aus, Instruktions-Pipelines lesen Befehle voraus
- Ressourcen-, Daten- und Kontrollflußkonflikte führen zu „Blasen“
- Zweigvorhersage reduziert Kontrollflußkonflikte in Instruktions-Pipelines
 - nach oben / nach unten
 - Delayed Branches: manuell optimieren
 - Branch History Table: Sprünge merken