Rechnerarchitektur, Foliensatz 7
Pipeline-Verarbeitung

G. Kemnitz

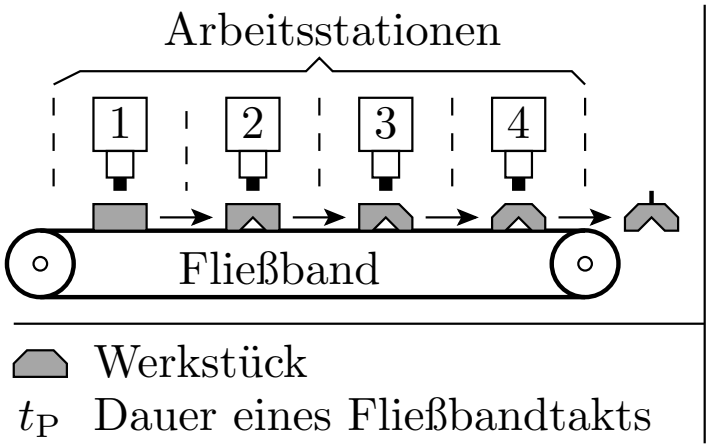
Institut für Informatik, TU Clausthal (RA-F7.pdf)

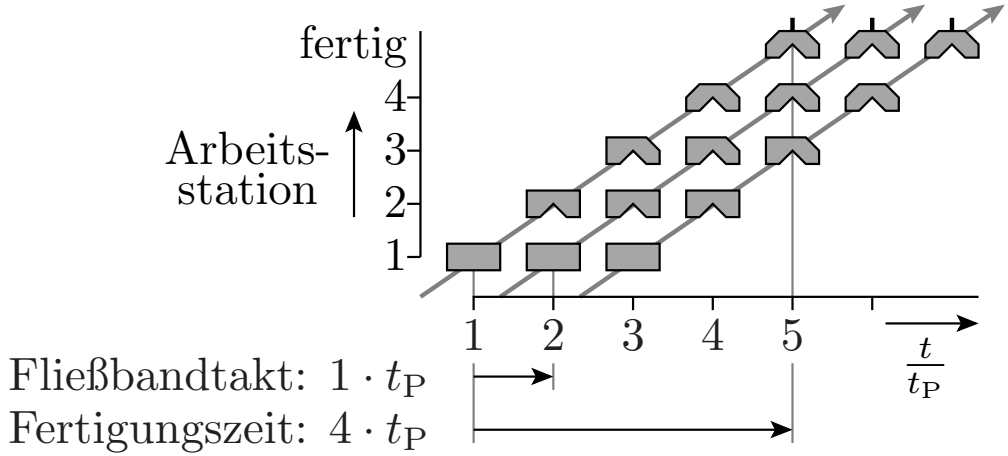
23. Januar 2020

Pipeline-Verarbeitung
1.1 MiPro mit Pipeline
1.2 Pipeline-Auslastung
1.3 Verarbeitungs-Pipeline
1.4 LS-Pipeline
1.5 Sprung-Pipeline
1.6 Unterprogramme

Pipeline-Verarbeitung

Pipeline-Verarbeitung

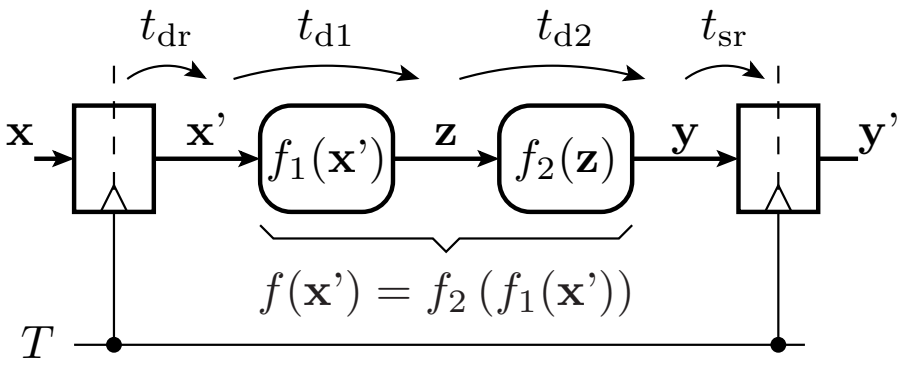


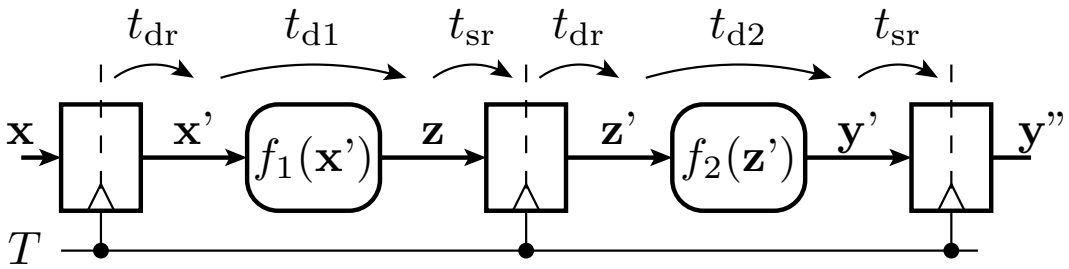


- Aufteilung einer Gesamtaufgabe in N_P Arbeitsschritte.
- Gesamtaufwand je Objekt: $N_P \cdot t_T$ (t_T – Periode Schritttakt)
- Je Takt wird ein Objekt fertig.

Parallelverarbeitung ohne viel Zusatzaufwand.

Angewandt auf Hardware





t_{dr}	Registerverzögerungszeit
t_{d1}	Verzögerungszeit von f_1
t_{d2}	Verzögerungszeit von f_2
t_{sr}	Registervorhaltezeit
$\mathbf{x}$	Eingabesignal
$\mathbf{z}$	Zwischensignal
$\mathbf{y}$	Ausgabesignal
...'	abgetastetes Signal

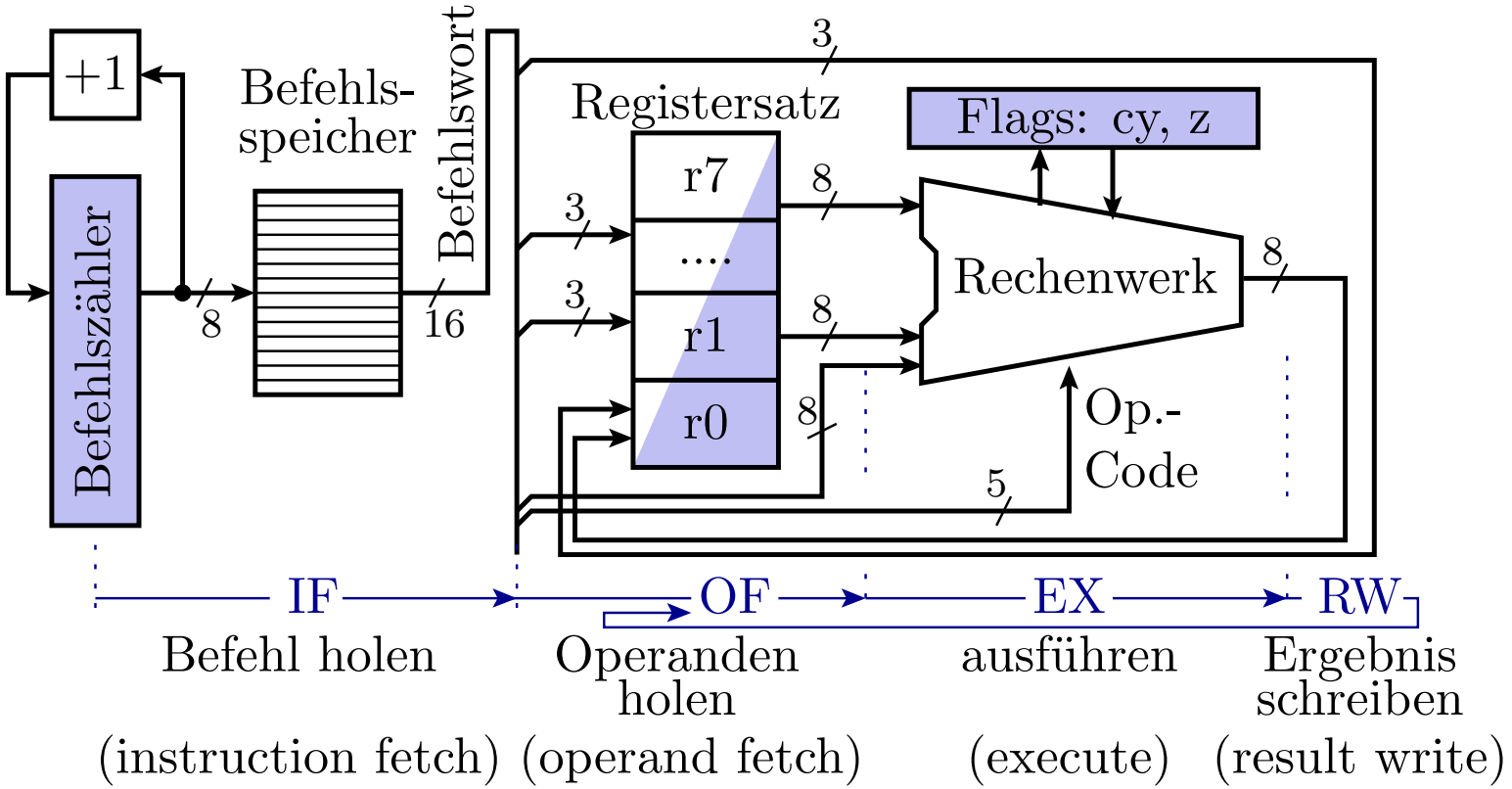
minimale Taktperiode

- ohne Pipeline
  $t_{dr} + t_{sr} + t_{d1} + t_{d2}$
- mit Pipeline
  $t_{dr} + t_{sr} + \max(t_{d1}, t_{d2})$

- Aufwand: ein zusätzliches Register je Pipeline-Stufe.
- Viel billiger als mehrfache Hardware. Vorzuglösung für Parallelverarbeitung.

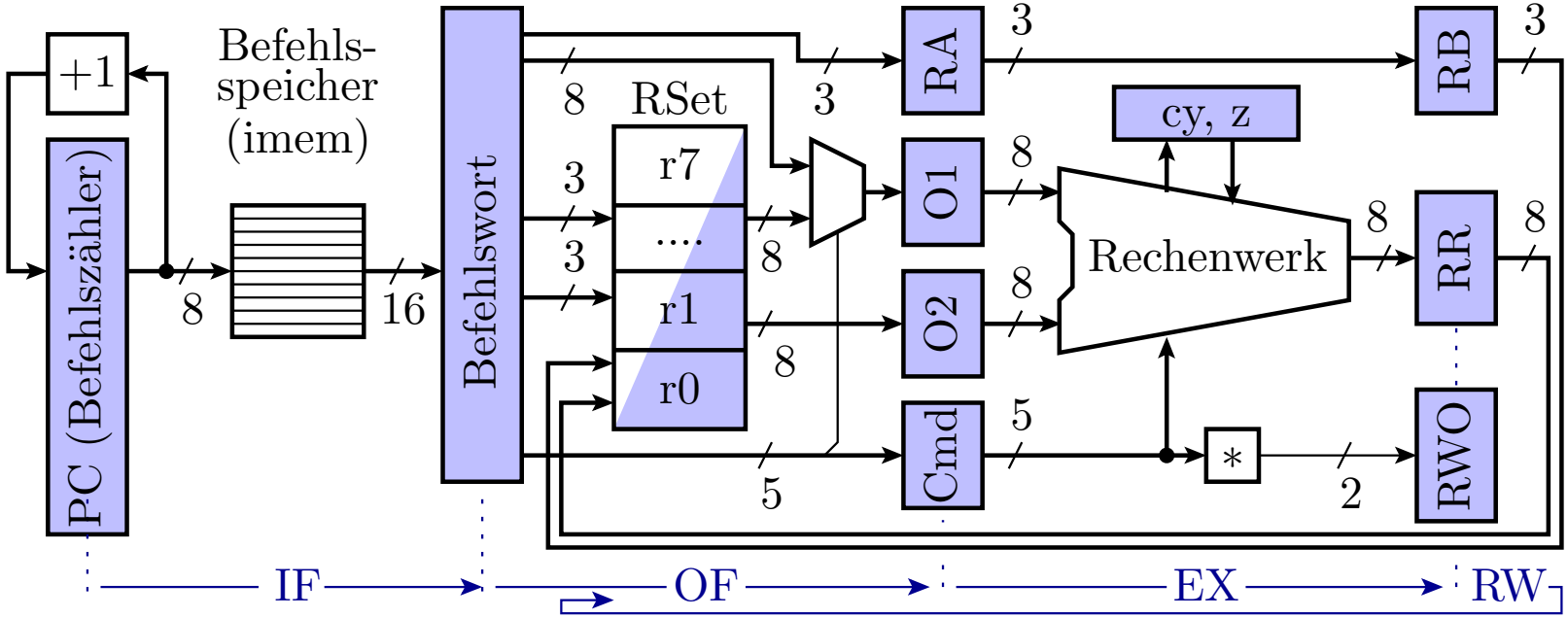
MiPro mit Pipeline

Der Kern des Minimalprozessors ohne Pipeline

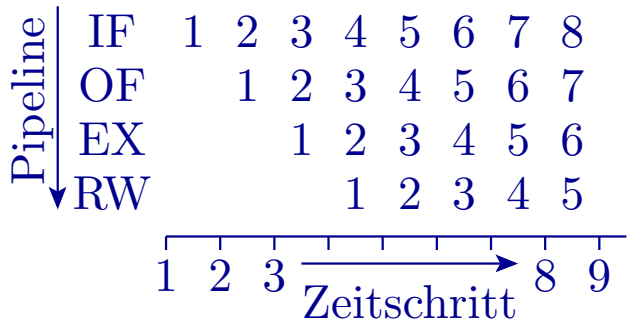


- Quellregister: Operanden- und Statusregister, Befehlszähler.
- Zielregister: Ergebnis- und Statusregister, Befehlszähler.
- Verarbeitungsschritte: Befehl holen, Operanden holen, ...

Aufteilung in Pipeline-Phasen



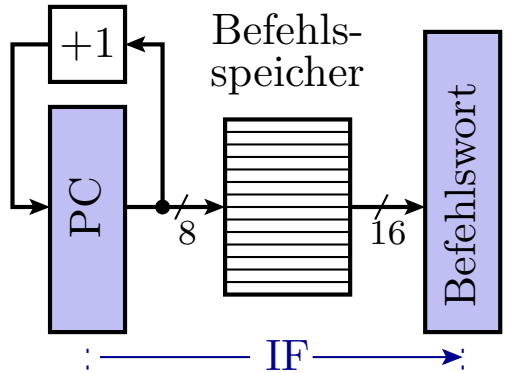
r0 bis r7	Arbeitsregister
O1, O2	Operandenregister
Cmd	Befehlsregister
RA, RB	Ergebnisadressreg.
RR	Ergebnisregister
RWO	OP-Code RW-Phase



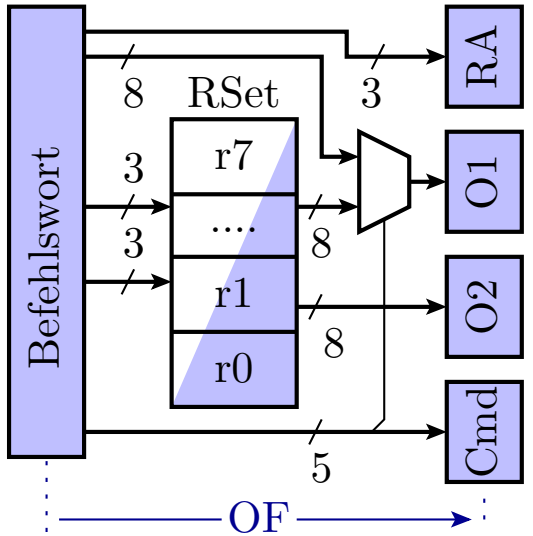
Die Aufteilung der Verarbeitungsschritte in Pipeline-Phasen erfolgt durch Einbau getakteter Register für die Zwischenergebnisse.

Operationen der einzelnen Pipeline-Phasen:

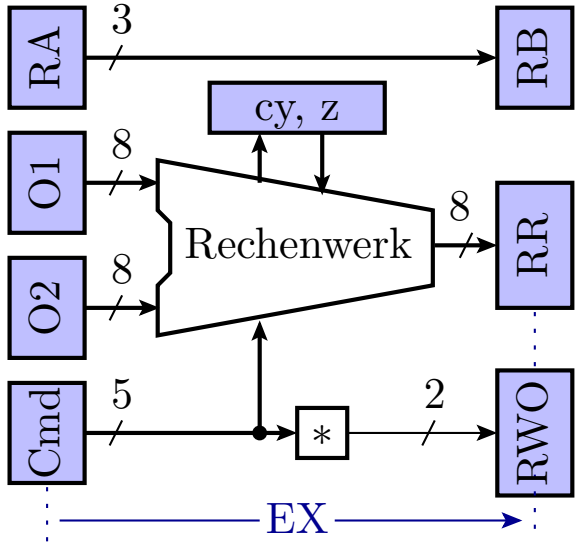
- IF (**I**nstruction **F**etch) Befehl holen: Adressierung des Befehlsspeichers vom Befehlszähler und Übernahme des Befehlswortes in das Befehlswortregister.



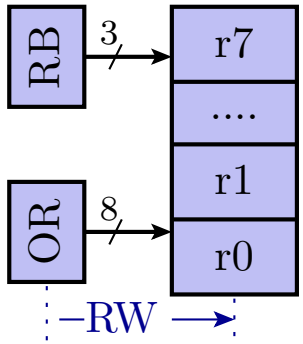
- OF (**O**perand **F**etch) Operanden holen: Adressierung des Registersatzes mit den Operandenadressen und Übernahme der Registerinhalte in die Operandenregister. Weitergabe der Ergebnisadresse und des Operationscodes an die nächste Pipeline-Phase.



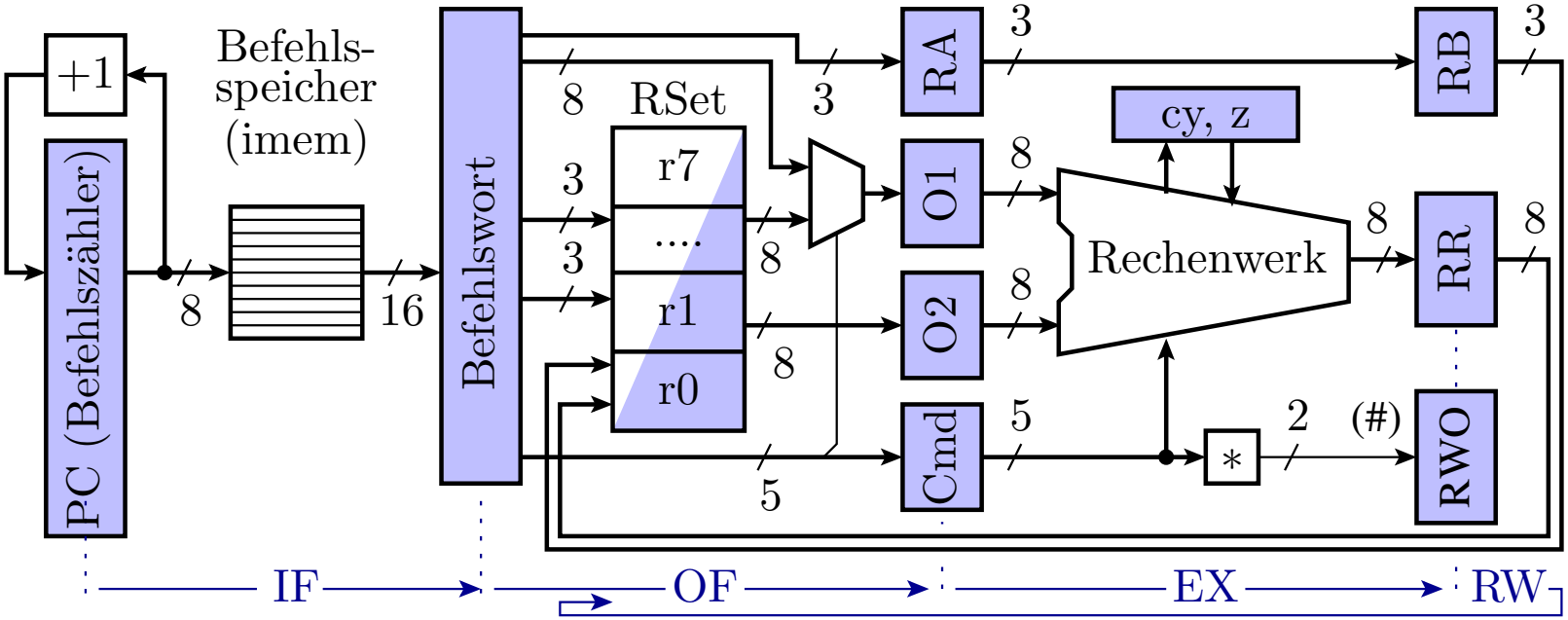
- EX (**Ex**ecute): Befehle ausführen: Aus den Operanden und dem Operationscode bildet das Rechenwerk das Ergebnis und Weitergabe der Ergebnisadresse und eines Operations-Codes RWO∈{-,R,L,S} an die RW-Pipeline-Phase.



- RW (**R**esult **W**rite) Ergebnis schreiben: Wenn (W)=R Adressierung des Registersatzes mit der Ergebnisadresse und Speichern des Ergebnisses.



Die Befehlsausführung dauert vier Schritte (Taktperioden). Es wird gleichzeitig an vier Befehlen gearbeitet. Der maximal erzielbare Verarbeitungsdurchsatz ist ein Befehl pro Schritt.



- Bei einer Aufteilung des Verarbeitungsflusses in mehrere gleichlange Pipeline-Phasen kann der Rechner wesentlich schneller getaktet werden und trotzdem in jedem Takt eine neue Operation beginnen und eine fertigstellen.
- Die Fertigstellung der einzelnen Befehle dauert mindestens genauso lange wie ohne Pipeline.
- Pipeline-Ergänzungen für LS-Befehle, Sprünge, ... folgen noch.

Pipeline-Auslastung

Pipeline-Auslastung

Verarbeitungsergebnisse werden erst zwei Takte nach Lesen der Operanden geschrieben. Verursacht Probleme. Beispiel:

Addition von vier Registerinhalten

Adr	Befehl
0	B1: addr r0,r0,r1
1	B2: addr r0,r0,r2
2	B3: addr r0,r0,r3

*	Register lesen
*n	lesen und mit n überschreiben
BW	Befehlswort

IF → OF → RW → EX

PC	BW	r_0	r_1	r_2	r_3	O1	O2	RR
0		7	2	11	17			
1	B1							
2	B2	*	*			7	2	
3	B3	*		*		7	11	9
4		*9			*	7	17	18
5		18						24
6		24						

In r_0 steht in Takt 5 $r_0 + r_1$, in Takt 6 $r_0 + r_2$ und in Takt 7 $r_0 + r_3$. Statt $r_0 + r_1 + r_2 + r_3$ wird $r_0 + r_3$, d.h. ein falsches Ergebnis berechnet.

Einfügen von noop[1]-Befehlen

```
Adr  Befehl
 0   B1: addr r0,r0,r1
 1   noop
 2   noop
 3   B2: addr r0,r0,r2
 4   noop
 5   noop
 6   B3: addr r0,r0,r3
```

*	Register lesen
*n	lesen und mit n überschreiben

PC	BW	r_0	r_1	r_2	r_3	O1	O2	RR
0		7	2	11	17			
1	B1							
2	noop							
3	noop							
4	B2							
5	noop							
6	noop							
7	B3							
8	noop							
9	noop							
10	noop							
11	noop							

- Wird die Summe der 4 Registerwerte richtig berechnet?
- Wie lange dauern jetzt die drei Additionen?

[1] noop – **No Op**eration, für den Beispielprozessor Op-Code 0x0000.

Lösung

Adr	Befehl
0	B1: addr r0,r0,r1
1	noop
2	noop
3	B2: addr r0,r0,r2
4	noop
5	noop
6	B3: addr r0,r0,r3

*	Register lesen
*n	lesen und mit n überschreiben

IF OF RW EX

PC	BW	r_0	r_1	r_2	r_3	O1	O2	RR
0		7	2	11	17			
1	B1							
2	noop	*	*			7	2	
3	noop							9
4	B2	9						
5	noop	*		*		9	11	
6	noop							20
7	B3	20						
8	noop	*			*	20	17	
9	noop							37
10	noop	37						
11	noop							

- Die Summe der 4 Registerwerte wird richtig berechnet
- Jede Additionen benötigen 3 Takte. Nicht schneller als ohne Pipeline.

Optimierte Berechnungsreihenfolge

```
0  B1: addr r0,r0,r1
1  B2: addr r2,r2,r3
2  noop
3  noop
4  B3: addr r0,r0,r2
```

*	Register lesen
*n	lesen und mit n überschreiben

IF → OF → RW → EX

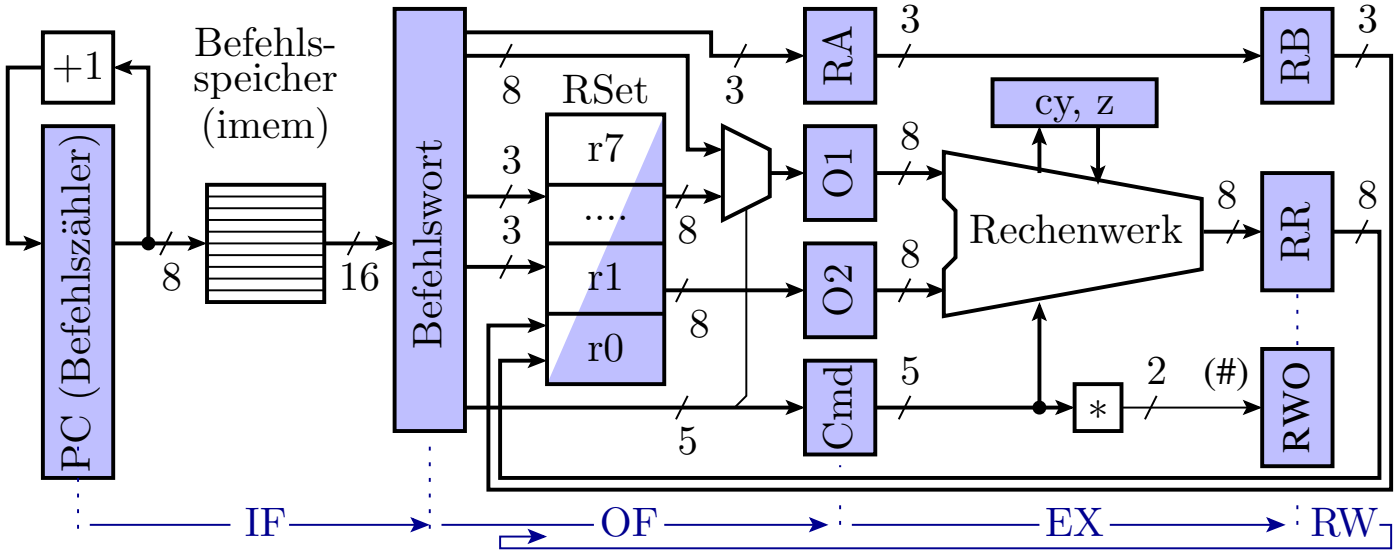
PC	BW	r_0	r_1	r_2	r_3	O1	O2	RR
0		7	2	11	17			
1	B1							
2	B2	*	*			7	2	
3	noop			*	*	11	17	9
4	noop	9						28
5	B3			28				
6	noop	*		*		9	28	
7	noop							37
8	noop	37						

Die Additionen $r_0 + r_1$ und $r_2 + r_3$ können direkt nacheinander erfolgen. Ausführungszeit zwei Takte weniger.

Die Abarbeitungszeit und Größe von Programmen hängen erheblich von der Compiler-Optimierung ab.

Verarbeitungs-Pipeline

Darstellung des Prozessorzustands



PC	Befehlswort	Cmd	O1	O2	RR	c	z	RA	RB	#	r0	r1	r2	r3
01	ld_i r1,4a,..	noop	00	00	00	0	0	r0	r0	–	00	00	00	00
02	ld_i r0,73,..	....	..	..	..	.	.	..	..	.	..	..	..	..

Register: Cmd – Operation; O1, O2 – Operanden; c, z – Flags; RR – Ergebnis; RA, RB – Ergebnisadresse; #∈{-, R, L, S} – RW-Operation; r0, r1, ... – Arbeitsregister.

Pipeline-Verarbeitung für Logikoperationen

Ergänzen Sie die Registerwerte.

PC	Befehlswort	Cmd	O1	O2	RR	c	z	RA	RB	#	r0	r1	r2	r3
01	ld_i r1,4a,..	noop	00	00	00	.	.	r0	r0	–	..	..	..	..
02	ld_i r0,73,..	....	..	..	..	.	.	..	..	.	..	..	..	..
03	noop .. ,.. ,..	....	..	..	..	.	.	..	..	.	..	..	..	..
04	move r3,r1,..	....	..	..	..	.	.	..	..	.	..	..	..	..
05	move r2,r0,..	....	..	..	..	.	.	..	..	.	..	..	..	..
06	noop .. ,.. ,..	....	..	..	..	.	.	..	..	.	..	..	..	..
07	andi r3,f0,..	....	..	..	..	.	.	..	..	.	..	..	..	..
08	andi r2,0f,..	....	..	..	..	.	.	..	..	.	..	..	..	..
09	noop .. ,.. ,..	....	..	..	..	.	.	..	..	.	..	..	..	..
0a	noop .. ,.. ,..	....	..	..	..	.	.	..	..	.	..	..	..	..
0b	or_r r4,r2,r3	....	..	..	..	.	.	..	..	.	..	..	..	..
0c	xorr r2,r3,r3	....	..	..	..	.	.	..	..	.	..	..	..	..
0d	noop .. ,.. ,..	....	..	..	..	.	.	..	..	.	..	..	..	..
0e	noop .. ,.. ,..	....	..	..	..	.	.	..	..	.	..	..	..	..

Lösung

PC	Befehlswort	Cmd	O1	O2	RR	c	z	RA	RB	#	r0	r1	r2	r3
01	ld_i r1,4a,..	noop	00	00	00	.	.	r0	r0	–	..	..	..	..
02	ld_i r0,73,..	ld_i	4a	00	**	.	.	r1	r0	–	..	..	..	..
03	noop ..,..,..	ld_i	73	00	4a	.	.	r0	r1	R	..	..	..	..
04	move r3,r1,..	noop	00	00	73	.	.	r0	r0	R	..	4a	..	..
05	move r2,r0,..	move	4a	00	**	.	.	r3	r0	–	73	**	..	..
06	noop ..,..,..	move	73	00	4a	.	.	r2	r3	R	**	**	..	..
07	andi r3,f0,..	noop	00	00	73	.	.	r0	r2	R	**	**	..	4a
08	andi r2,0f,..	andi	f0	4a	**	.	.	r3	r0	–	**	**	73	**
09	noop ..,..,..	andi	0f	73	40	.	0	r2	r3	R	**	**	**	**
0a	noop ..,..,..	noop	00	00	03	.	0	r0	r2	R	**	**	**	40
0b	or_r r4,r2,r3	noop	00	00	**	.	*	r0	r0	–	**	**	03	**
0c	xorr r2,r3,r3	or_r	03	40	**	.	*	r4	r0	–	**	**	**	**
0d	noop ..,..,..	xorr	40	40	43	.	0	r5	r4	R	**	**	**	**
0e	noop ..,..,..	noop	00	00	00	.	1	r0	r5	R	**	**	**	**

Cmd – Operationscode; O1, O2 – Operandenregister; c, z – Flags; RA, RB – Ergebnisadr.; RWO/#∈{-, R, L, S} – RW-Operation; r0, r1, ... – Arbeitsregister.

Beispielaufgabe 2-Byte-Addition

```
r0:r1 := 733A; r2:r3 := 13E7
r4:r5 := r0:r1 + r2:r3     (Ergebnis: 8721)
r4:r5 := 8623 - r4:r5      (Ergebnis: FF02)
```

PC	Befehlswort	O1	O2	RR	c	z	RA	RB	#	r0	r1	r2	r3	r4	r5
01	ld_i r1,3a,..	00	00	00	.	.	r0	r0	–	..	..	..	..	..	..
02	ld_i r0,73,..	..	..	..	.	.	..	..	.	..	..	..	..	..	..
03	ld_i r3,e7,..	..	..	..	.	.	..	..	.	..	..	..	..	..	..
04	ld_i r2,13,..	..	..	..	.	.	..	..	.	..	..	..	..	..	..
05	noop .. ,.. ,..	..	..	..	.	.	..	..	.	..	..	..	..	..	..
06	addr r5,r1,r3	..	..	..	.	.	..	..	.	..	..	..	..	..	..
07	adcr r4,r0,r2	..	..	..	.	.	..	..	.	..	..	..	..	..	..
08	noop .. ,.. ,..	..	..	..	.	.	..	..	.	..	..	..	..	..	..
09	subi r5,23,..	..	..	..	.	.	..	..	.	..	..	..	..	..	..
0a	sbci r4,86,..	..	..	..	.	.	..	..	.	..	..	..	..	..	..
0b	noop .. ,.. ,..	..	..	..	.	.	..	..	.	..	..	..	..	..	..
0c	noop .. ,.. ,..	..	..	..	.	.	..	..	.	..	..	..	..	..	..
0d	noop .. ,.. ,..	..	..	..	.	.	..	..	.	..	..	..	..	..	..

Lösung

```
r0:r1 := 733A; r2:r3 := 13E7
r4:r5 := r0:r1 + r2:r3     (Ergebnis: 8721)
r4:r5 := 8623 - r4:r5      (Ergebnis: FF02)
```

PC	Befehlswort	O1	O2	RR	c	z	RA	RB	#	r0	r1	r2	r3	r4	r5
01	ld_i r1,3a,..	00	00	00	.	.	r0	r0	–	..	..	..	..	..	..
02	ld_i r0,73,..	3a	00	**	.	.	r1	r0	–	..	..	..	..	..	..
03	ld_i r3,e7,..	73	00	3a	.	.	r0	r1	R	..	..	..	..	..	..
04	ld_i r2,13,..	e7	00	73	.	.	r3	r0	R	..	3a	..	..	..	..
05	noop .. ,.. ,..	13	00	e7	.	.	r2	r3	R	73	**	..	..	..	..
06	addr r5,r1,r3	00	00	13	.	.	r0	r2	R	**	**	..	e7	..	..
07	adcr r4,r0,r2	3a	e7	**	.	.	r5	r0	–	**	**	13	**	..	..
08	noop .. ,.. ,..	73	13	21	1	0	r4	r5	R	**	**	**	**	..	..
09	subi r5,23,..	00	00	87	0	0	r0	r4	R	**	**	**	**	..	21
0a	sbci r4,86,..	23	21	**	*	*	r5	r0	–	**	**	**	**	87	**
0b	noop .. ,.. ,..	86	87	02	0	0	r4	r5	R	**	**	**	**	**	**
0c	noop .. ,.. ,..	00	00	ff	1	0	r0	r4	R	**	**	**	**	**	02
0d	noop .. ,.. ,..	00	00	**	*	*	r0	r0	–	**	**	**	**	ff	**

LS-Pipeline

Lade- / Speicherbefehle des Minimalprozessors

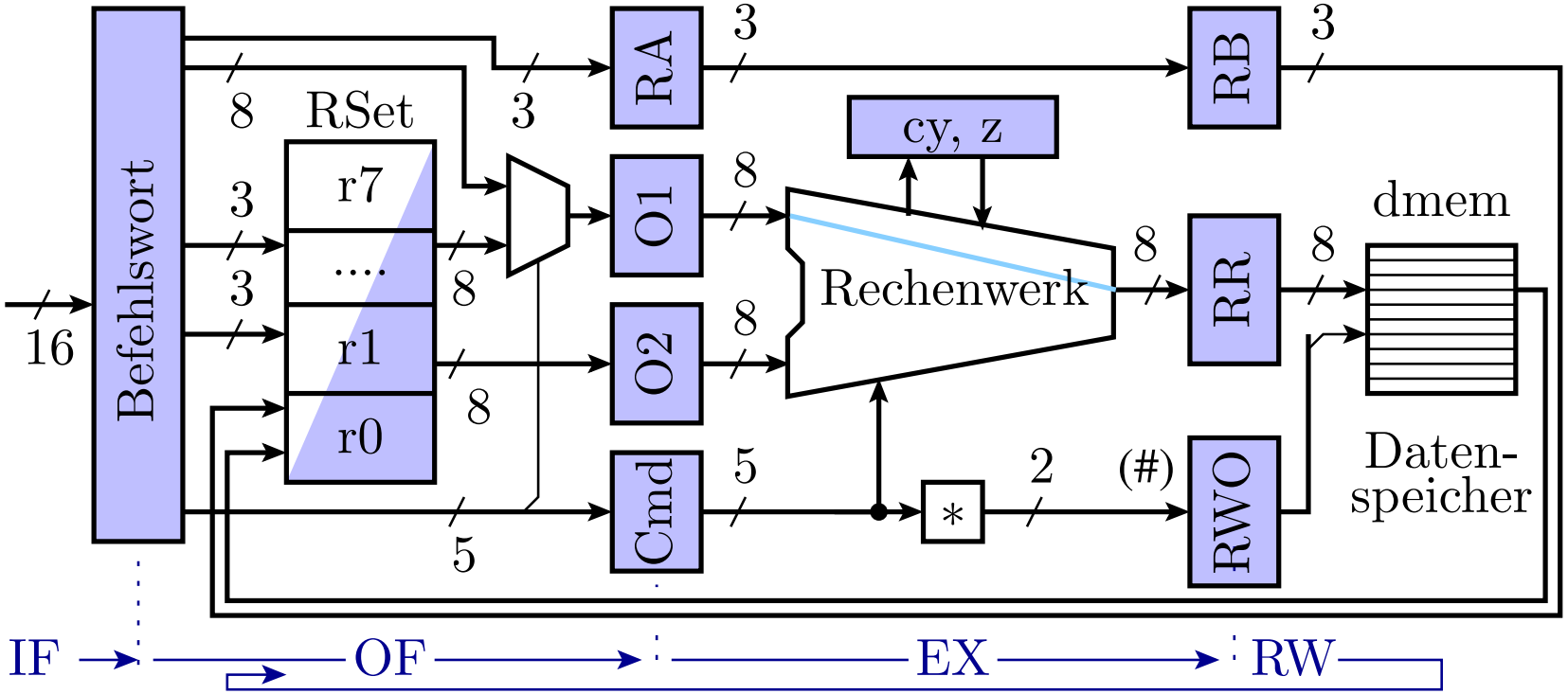
Teilbitvektor	15 ... 11	10 9 8	7 6 5	4 3 2	1 0
cmd rd,ra	cnr	rd	ra		
cmd rd,imm	cnr	rd	imm		

Befehl	Operation	Flags	cnr
load rd,imm	rd := *(imm)		3
stor rd,imm	*(imm) := rd		4
ld_r rd,ra	rd := *(ra)		17
st_r rd,ra	*(ra) := rd		18

(*(...) Speicherinhalt von). Unterstütze Adressierungsarten:

- direkt für die Adressierung von Variablen mit festen Adressen.
- indirekt für die Adressierung mit berechneten Adressen.

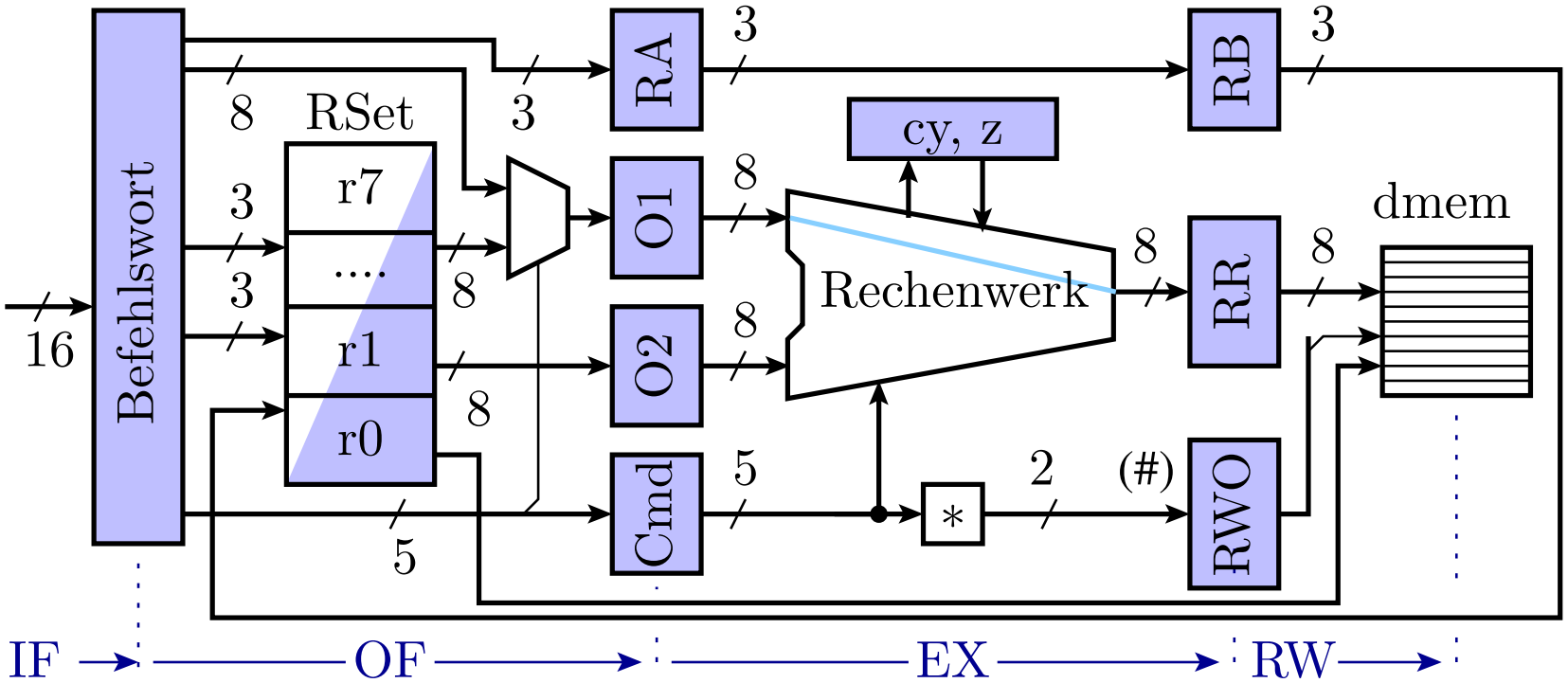
Lade-Pipeline



* Bildung des 2-Bit-Operationswortes für die RW-Phase

- EX-Phase: Adressrechnung, Weitergabe RWO-Code für Laden.
- RW-Phase: Laden des Ergebnisregisters mit Speicherinhalt, gesteuert durch RWO=L.

Speicher-Pipeline



- EX-Phase: Adressrechnung, Weitergabe RWO-Code Speichern.
- RW-Phase: Kopieren Ergebnisregisterinhalt in den Speicher, gesteuert durch RWO=S.

Testbeispiel mit Lade- und Speicherbefehlen

```
*(5) := 0x48;
r1 := 6; *(r1)= 0x31;
r3 := *(5);
r4 := *(r1);
```

Programmiert für Pipeline-Verarbeitung:

```
0000: ld_i r0,48,..; r0 := 0x48
0001: stor r0,05,..; *(5) := r0
0002: ld_i r1,06,..; r1 := 0x06 (Adresse)
0003: noop ..,..,..
0004: ld_i r2,31,..; r2 := 0x31 (Daten)
0005: noop ..,..,..
0006: st_r r2,r1,..; *(r1) := r2
0007: load r3,05,..; r3 := *(5)
0008: ld_r r4,r1,..; r4 := *(r1)
0009: noop ..,..,..
```

Aufgabe

Ergänzen Sie die Register- und Datenspeicherwerte.

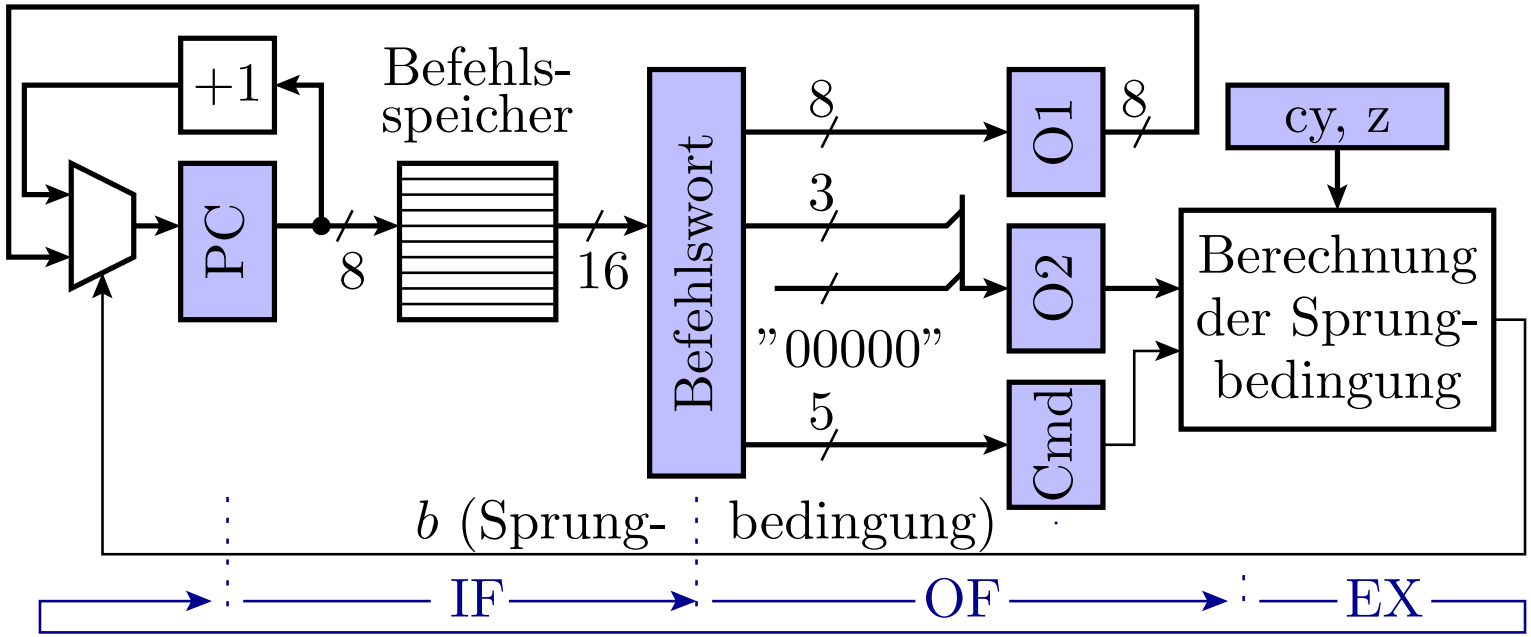
PC	Befehlswort	Cmd	O1	O2	RR	c	z	RA	RB	#	r0	r1	r2	r3	r4
01	ld_i r0,48,..	noop	00	00	00	.	.	r0	r0	–	..	..	..	..	..
02	stor r0,05,..	....	..	..	..	.	.	..	..	.	..	..	..	..	..
03	ld_i r1,06,..	....	..	..	..	.	.	..	..	.	..	..	..	..	..
04	noop .. ,.. ,..	....	..	..	..	.	.	..	..	.	..	..	..	..	..
05	ld_i r2,31,..	....	..	..	..	.	.	..	..	.	..	..	..	..	..
		;dmem(0:7) = [.. .. ..]													
06	noop .. ,.. ,..	....	..	..	..	.	.	..	..	.	..	..	..	..	..
07	st_r r2,r1,..	....	..	..	..	.	.	..	..	.	..	..	..	..	..
08	load r3,05,..	....	..	..	..	.	.	..	..	.	..	..	..	..	..
09	ld_r r4,r1,..	....	..	..	..	.	.	..	..	.	..	..	..	..	..
0a	noop .. ,.. ,..	....	..	..	..	.	.	..	..	.	..	..	..	..	..
		;dmem(0:7) = [.. .. ..]													
0b	noop .. ,.. ,..	....	..	..	..	.	.	..	..	.	..	..	..	..	..
0c	noop .. ,.. ,..	....	..	..	..	.	.	..	..	.	..	..	..	..	..

Lösung

PC	Befehlswort	Cmd	O1	O2	RR	c	z	RA	RB	#	r0	r1	r2	r3	r4
01	ld_i r0,48,..	noop	00	00	00	.	.	r0	r0	–	..	..	..	..	..
02	stor r0,05,..	ld_i	48	00	**	.	.	r0	r0	–	..	..	..	..	..
03	ld_i r1,06,..	stor	05	00	48	.	.	r0	r0	R	..	..	..	..	..
04	noop .. ,.. ,..	ld_i	06	00	05	.	.	r1	r0	S	48	..	..	..	..
05	ld_i r2,31,..	noop	00	00	06	.	.	r0	r1	R	**	..	..	..	..
		;dmem(0:7) = [.. .. .. 48 .. ..]													
06	noop .. ,.. ,..	ld_i	31	00	06	.	.	r2	r0	–	**	06	..	..	..
07	st_r r2,r1,..	noop	00	00	31	.	.	r0	r2	R	**	**	..	..	..
08	load r3,05,..	st_r	06	00	**	.	.	r2	r0	–	**	**	31	..	..
09	ld_r r4,r1,..	load	05	00	06	.	.	r3	r2	S	**	**	**	..	..
0a	noop .. ,.. ,..	ld_r	06	00	05	.	.	r4	r3	L	**	**	**	..	..
		;dmem(0:7) = [.. .. .. ** 31 ..]													
0b	noop .. ,.. ,..	noop	00	00	06	.	.	r0	r4	L	**	**	**	48	..
0c	noop .. ,.. ,..	noop	00	00	**	.	.	r0	r0	–	**	**	**	**	31

Sprung-Pipeline

Sprung-Pipeline




```
jump imm,cond; if (b) pc := imm; else pc++;
```

- OF-Phase: Sprungziel in O1 und Sprungbedingung in O2 laden.
- EX-Phase: bedingte Übernahme von O1 in den Befehlszähler.
- RW-Phase: keine Operation.

Beispielprogramm mit einer Schleife

```
    r0 := 1; r1 := 34;
M:  *(r0) := r1;
    r1 := r1 - r0;
    r0 := r0 + 1;
    wenn r0 <= 3 springe zu M; (3 Schleifendurchläufe)
```

Programmiert für Pipeline-Verarbeitung:

```
0x00: ld_i r0,01,..  ; r0 := 0x01
0x01: ld_i r1,34,..  ; r1 := 0x34
0x02: noop ..,..,..  ; warte, bis r0 geladen ist
0x03: comp r0,03,..  ; vergleiche r0 mit 0x3
0x04: st_r r1,r0,..  ; *(r0) := r1;
0x05: jump 02,lth..  ; wenn r0<0x3 springe zu 0x2
0x06: addi r0,01,..  ; r0 := r0+1  (Delayslot 1)
0x07: subr r1,r1,r0  ; r1 := r1+r0 (Delayslot 2)
0x08: noop ..,..,..  ; 1. Anw. nach der Schleife
```

Programmabarbeitung

```
PC|  Befehlswort  |Cmd  |O1|O2|RR|c|z|RA|RB|#|r0 r1 r2 r3 r4
01|ld_i r0,01,..|noop |00|00|00|0|0|r0|r0|-|.. .. .. .. ..
; 1. Schleifendurchlauf:
02|ld_i r1,34,..|ld_i |01|00|**|.|.|r0|r0|-|.. .. .. .. ..
03|noop ..,..,..|ld_i |34|00|01|.|.|r1|r0|R|.. .. .. .. ..
04|comp r0,03,..|noop |00|00|34|.|.|r0|r1|R|01 .. .. .. ..
05|st_r r1,r0,..|comp |03|01|**|.|.|r0|r0|-|** 34 .. .. ..
06|jump 02,lth..|st_r |01|00|02|0|0|r1|r0|-|** ** .. .. ..
07|addi r0,01,..|jump |02|07|01|*|*|r7|r1|S|** ** .. .. ..
; 2. Schleifendurchlauf:
02|subr r1,r1,r0|addi |01|01|01|*|*|r0|r7|-|** ** .. .. ..
; Abschluss 1. Schreibop.: dmem(0:7)=[.. 34 .. .. .. .. ..
03|noop ..,..,..|subr |34|01|02|0|0|r1|r0|R|** ** .. .. ..
04|comp r0,03,..|noop |00|00|33|0|0|r0|r1|R|02 ** .. .. ..
05|st_r r1,r0,..|comp |03|02|**|*|*|r0|r0|-|** 33 .. .. ..
```

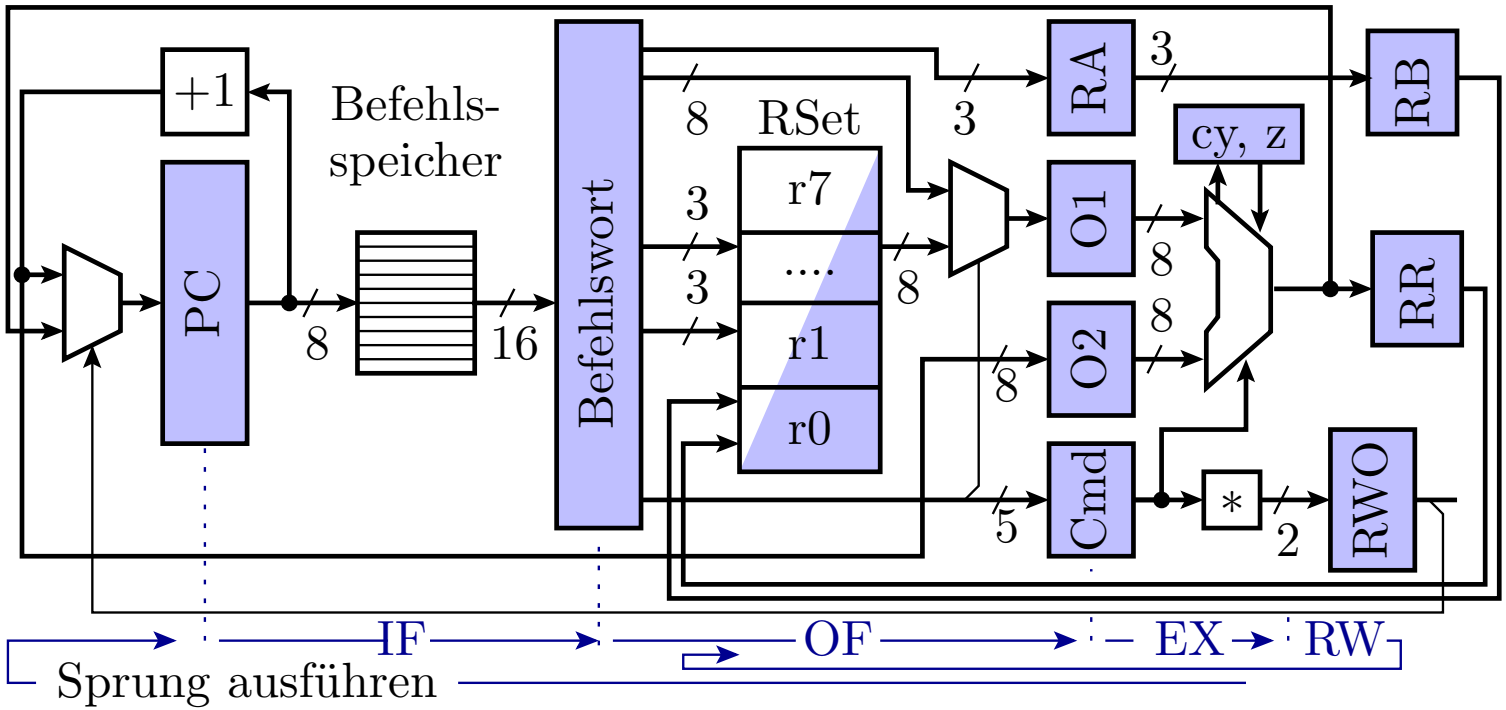
Fortsetzung der Programmabarbeitung

```
PC|  Befehlswort  |Cmd  |O1|O2|RR|c|z|RA|RB|#|r0 r1 r2 r3 r4
05|  Fortsetzung nach Laden des Sprungbefehls
06|jump 02,lth..|st_r|02|00|01|0|0|r1|r0|-|02 33 .. .. ..
07|addi r0,01,..|jump|02|07|02|*|*|r7|r1|S|** ** .. .. ..
; 3. Schleifendurchlauf:
02|subr r1,r1,r0|addi|01|02|02|*|*|r0|r7|-|** ** .. .. ..
; Abschluss 2. Schreibop.: dmem(0:7)=[.. 34 33 .. .. .. ..
03|noop ..,..,..|subr|33|02|03|0|0|r1|r0|R|** ** .. .. ..
04|comp r0,03,..|noop|00|00|31|0|0|r0|r1|R|03 ** .. .. ..
05|st_r r1,r0,..|comp|03|03|**|*|*|r0|r0|-|** 31 .. .. ..
06|jump 02,lth..|st_r|03|00|00|0|1|r1|r0|-|** ** .. .. ..
07|addi r0,01,..|jump|02|07|03|*|*|r7|r1|S|** ** .. .. ..
08|subr r1,r1,r0|addi|01|03|03|*|*|r0|r7|-|** ** .. .. ..
; Abschluss 3. Schreibop.: dmem[0:7]=[.. 34 33 31 .. .. ..
09|noop ..,..,..|subr|31|03|04|0|0|r1|r0|R|** ** .. .. ..
0a|noop ..,..,..|noop|00|00|2e|0|0|r0|r1|R|04 ** .. .. ..
```

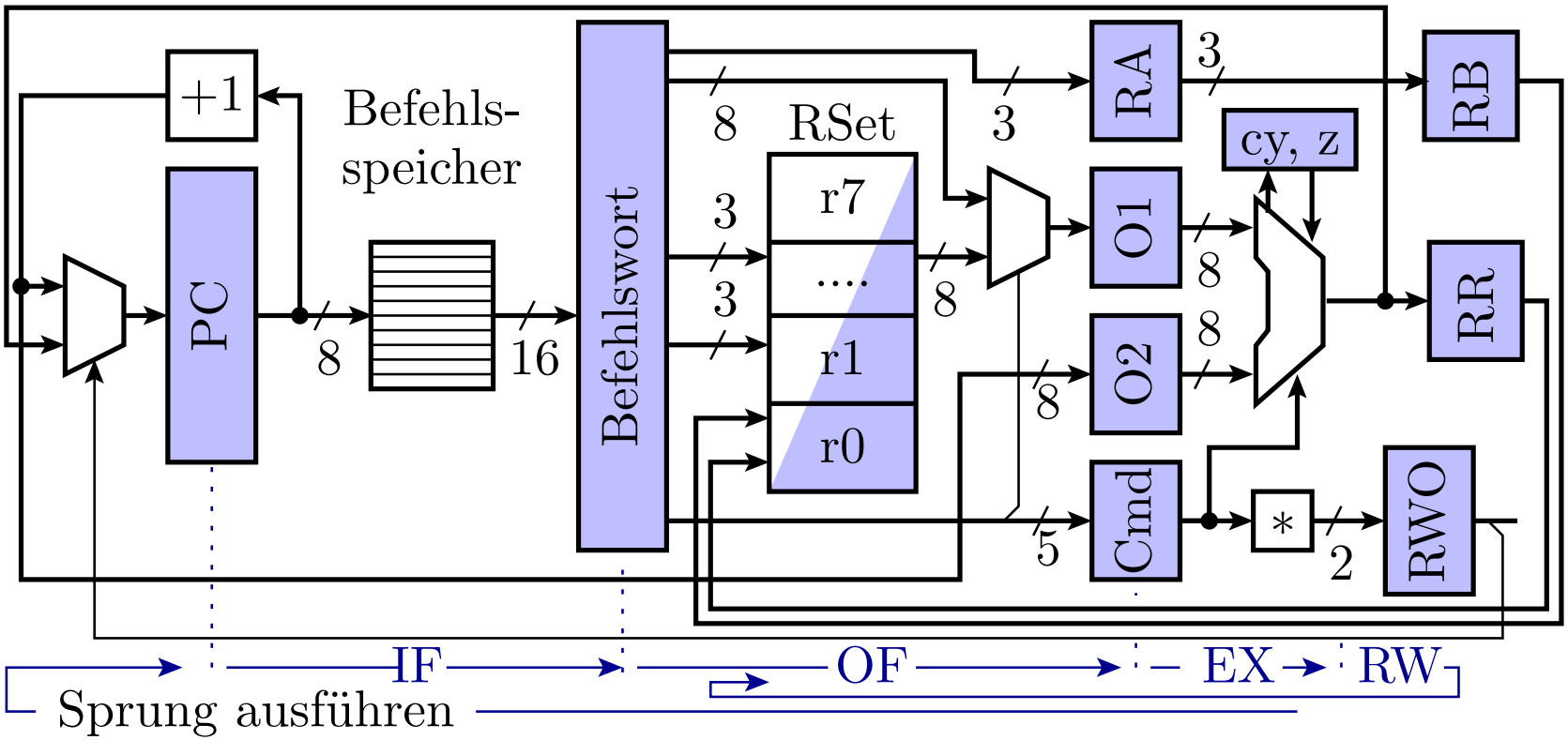
Unterprogramme

Unterprogrammaufruf und Rücksprung

Ein Unterprogrammaufruf speichert die Rückkehradresse in einem Register und der Rücksprung liest das Sprungziel aus einem Register.



call rd ,imm	rd:=pc+1, pc:=imm
retu rd	pc := rd



- OF-Phase: Direktwert (call) oder Registerinhalt (retu), in O1. Befehlszähler+1 in O2, Zieladresse in RA (nur retu).
- EX-Phase: Sprungausführung. Rückkehradresse und Registeradresse weiterreichen (nur call).
- RW-Phase: Rücksprungadresse in Register speichern (nur call).

Unterprogrammaufrufe mit Pipeline

Das nachfolgende Unterprogramm bekommt in dmem(1) einen Wert und in r1 eine Adresse übergeben und schreibt den übergebenen Wert + 0x13 in den Datenspeicher auf die Übergabeadresse:

```
0000: ld_i r0,35,..          Unterprogramm:
0001: stor r0,01,..          0010: load r3,01,..
0002: ld_i r1,02,..          0013: addi r3,13,..
0003: call r5,10,..          0014: st_r r3,r1,..
0006: ld_i r0,46,..          0015: retu r5,..,..
0007: stor r0,01,..
0008: ld_i r1,04,..          (Restliche Befehle noop)
0009: call r5,10,..
000c: jump 08,alw..  ; Endlosschleife
```

Testbeispiele:

- Aufruf mit *(1)=0x35 und r1=2, Ergebnis *(2)=0x48
- Aufruf mit *(1)=0x46 und r1=4, Ergebnis *(4)=0x59

```
PC|  Befehlswort  |Cmd  |O1|O2|RR|RA|RB|#|r0 r1 r2 r3 r4 r5
01|ld_i r0,35,..|noop |00|00|00|r0|r0|-|..  ..  ..  ..  ..  ..
02|stor r0,01,..|ld_i |35|00|**|r0|r0|-|..  ..  ..  ..  ..  ..
03|ld_i r1,02,..|stor |01|00|35|r0|r0|R|..  ..  ..  ..  ..  ..
; Unterprogrammaufruf mit dmem(1)=0x35 und r1=2
04|call r5,10,..|ld_i |02|00|01|r1|r0|S|35  ..  ..  ..  ..  ..
05|noop ..,..,..|call |10|04|02|r5|r1|R|**  ..  ..  ..  ..  ..
; Abschluss Schreibop. Aufruf 1: dmem=[..  35  ..  ..  ..  ..]
10|noop ..,..,..|noop |00|00|04|r0|r5|R|**  02  ..  ..  ..  ..
11|load r3,01,..|noop |00|00|**|r0|r0|-|**  **  ..  ..  ..  04
12|noop ..,..,..|load |01|00|**|r3|r0|-|**  **  ..  ..  ..  **
13|noop ..,..,..|noop |00|00|01|r0|r3|L|**  **  ..  ..  ..  **
14|addi r3,13,..|noop |00|00|**|r0|r0|-|**  **  ..  35  ..  **
15|st_r r3,r1,..|addi |13|35|**|r3|r0|-|**  **  ..  **  ..  **
16|retu r5,..,..|st_r |02|00|48|r3|r3|R|**  **  ..  **  ..  **
17|noop ..,..,..|retu |04|00|02|r5|r3|S|**  **  ..  48  ..  **
04|noop ..,..,..|noop |00|00|**|r0|r5|-|**  **  ..  **  ..  **
; Abschluss Schreibop. UP1: dmem=[..  35  48  ..  ..  ..  ..]
```

```
PC|  Befehlswort  |Cmd  |O1|O2|RR|RA|RB|#|r0 r1 r2 r3 r4 r5
; ... Abschluss Schreibop. UP1:
05|noop  .. ,.. ,.. |noop |00|00|**|r0|r0|-|35 02 .. 48 .. 04
06|noop  .. ,.. ,.. |noop |00|00|**|r0|r0|-|** ** .. ** .. **
07|ld_i  r0,46 ,.. |noop |00|00|**|r0|r0|-|** ** .. ** .. **
08|stor  r0,01 ,.. |ld_i |46|00|**|r0|r0|-|** ** .. ** .. **
09|ld_i  r1,04 ,.. |stor |01|00|46|r0|r0|R|** ** .. ** .. **
; Unterprogrammaufruf mit dmem(1)=0x46 und r1=4
0a|call  r5,10 ,.. |ld_i |04|00|01|r1|r0|S|46 ** .. ** .. **
0b|noop  .. ,.. ,.. |call |10|0a|04|r5|r1|R|** ** .. ** .. **
; Abschluss Schreibop. Aufruf 2: dmem=[.. 46 48 .. .. ..]
10|noop  .. ,.. ,.. |noop |00|00|0a|r0|r5|R|** 04 .. ** .. **
11|load  r3,01 ,.. |noop |00|00|**|r0|r0|-|** ** .. ** .. 0a
12|noop  .. ,.. ,.. |load |01|00|**|r3|r0|-|** ** .. ** .. **
13|noop  .. ,.. ,.. |noop |00|00|01|r0|r3|L|** ** .. ** .. **
14|addi  r3,13 ,.. |noop |00|00|**|r0|r0|-|** ** .. 46 .. **
15|st_r  r3,r1 ,.. |addi |13|46|**|r3|r0|-|** ** .. ** .. **
16|retu  r5 ,.. ,.. |st_r |04|00|59|r3|r3|R|** ** .. ** .. **
; ...
```

```
PC|  Befehlswort  |Cmd  |O1|O2|RR|RA|RB|W| r0 r1 r2 r3 r4 r5
...
17|noop  .. ,.. ,.. |retu |0a|00|04|r5|r3|S|46 04 .. 59 .. 0a
0a|noop  .. ,.. ,.. |noop |00|00|**|r0|r5|-|** ** .. ** .. **
Abschluss Schreibop. UP2: dmem = [.. 46 48 .. 59 .. ..]
0b|noop  .. ,.. ,.. |noop |00|00|**|r0|r0|-|** ** .. ** .. **
0c|noop  .. ,.. ,.. |noop |00|00|**|r0|r0|-|** ** .. ** .. **
Endlosschleife mit 2 Delay-Slots
0d|jump  0c,alw.. |noop |00|00|**|r0|r0|-|** ** .. ** .. **
0e|noop  .. ,.. ,.. |jump |0c|01|**|r1|r0|-|** ** .. ** .. **
0c|noop  .. ,.. ,.. |noop |00|00|**|r0|r1|-|** ** .. ** .. **
0d|jump  0c,alw.. |noop |00|00|**|r0|r0|-|** ** .. ** .. **
```