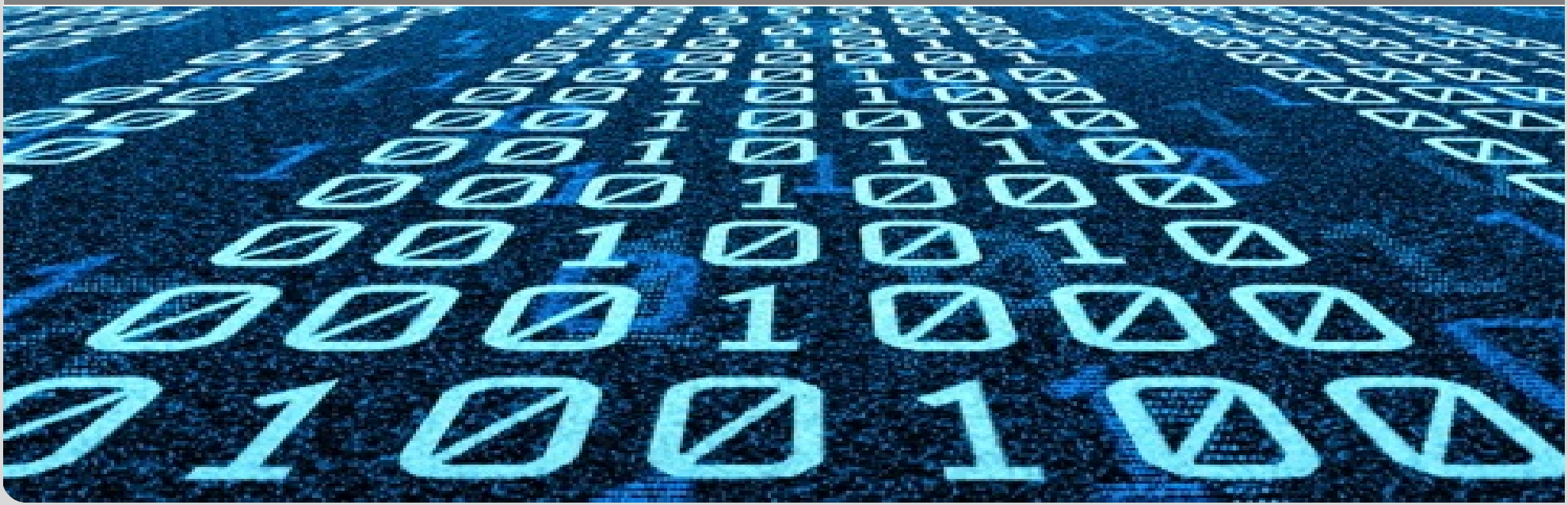
Karlsruher Institut für Technologie

8. Tutorium Wiederholung

Rechnerorganisation, Tutorium #13

Patrick Röper | 7. Januar 2020

FAKULTÄT FÜR INFORMATIK

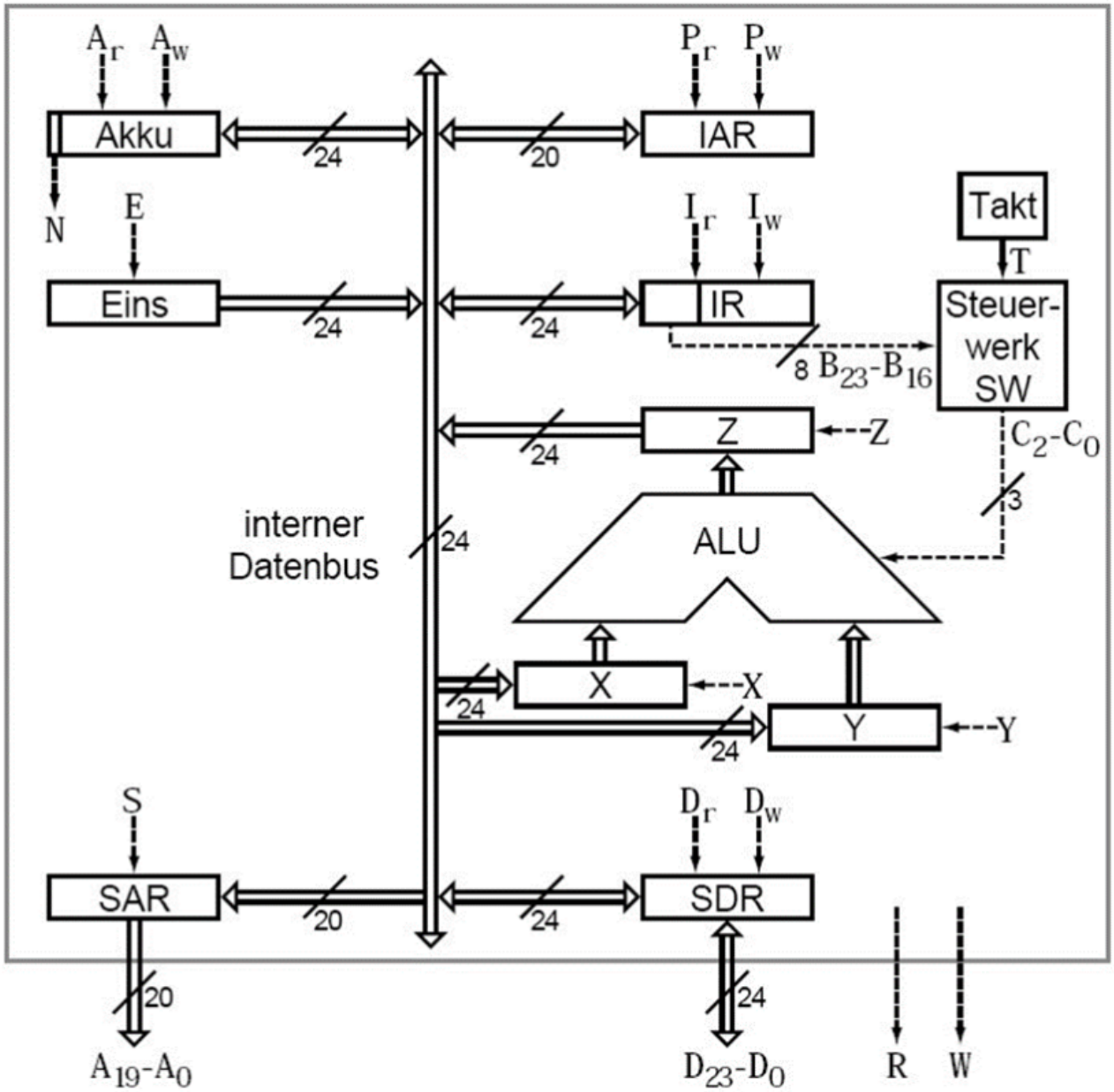


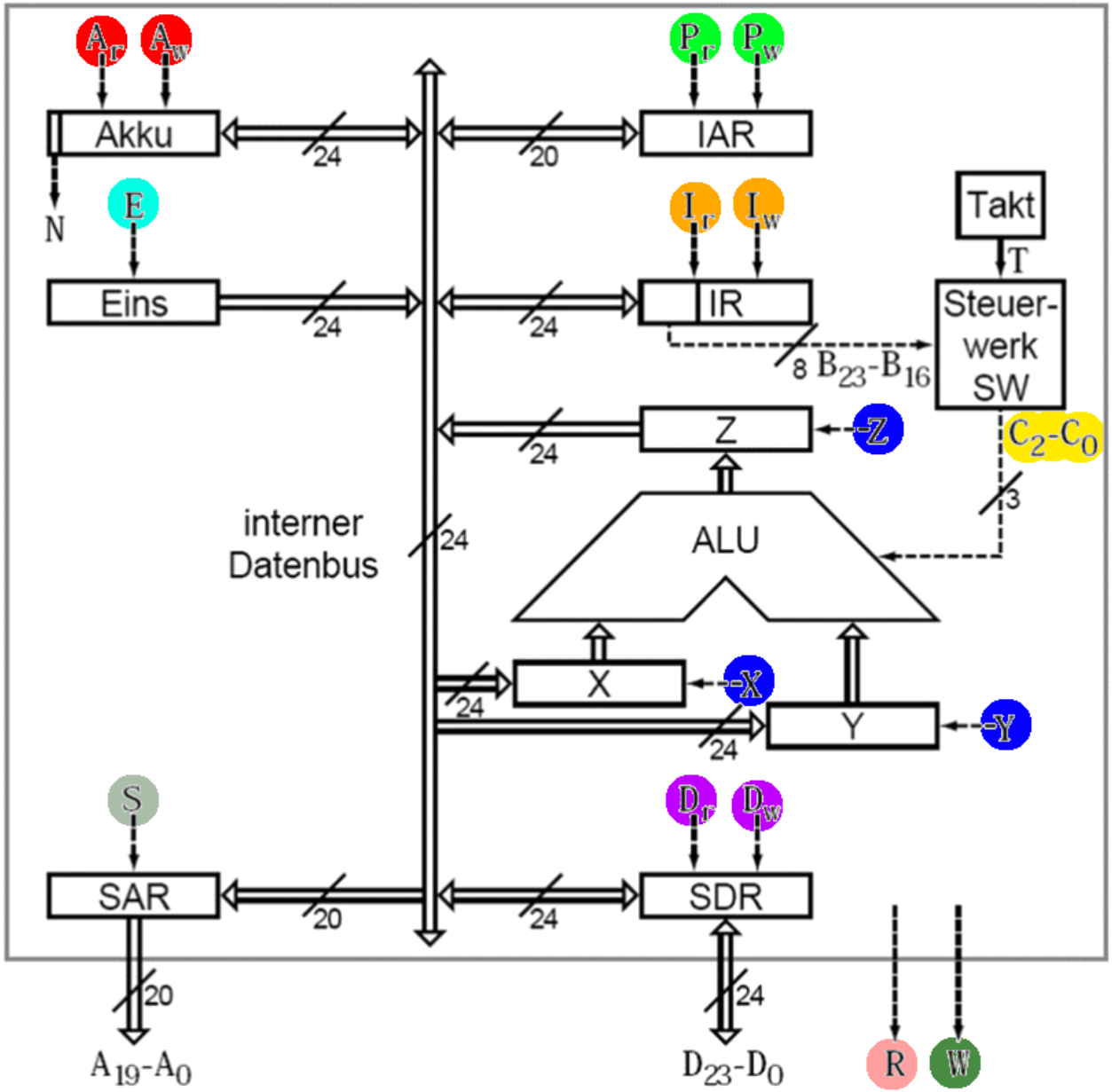
Roadmap

Klausurstruktur

- 5 Aufgaben aus DT
  - Schaltfunktion
  - Schaltnetze
  - Schaltwerke
  - Laufzeiteffekte
  - Rechnerarithmetik
- 5 Aufgaben aus RO
  - MIPS-Assembler
  - Pipeline
  - MIMA
  - Cache
  - Speicherverwaltung

Roadmap


A_r A_w
Akku
24
20
P_r P_w
IAR
N
E
Eins
24
24
I_r I_w
IR
Takt
T
Steuer-
werk
SW
8 B_{23}-B_{16}
Z
24
-Z
C_2-C_0
3
interner
Datenbus
24
ALU
X
-X
24
Y
24
-Y
S
SAR
20
20
D_r D_w
SDR
24
24
A_{19}-A_0
D_{23}-D_0
R
W


A_r
A_w
Akku
24
20
P_r
P_w
IAR
N
E
Eins
24
24
I_r
I_w
IR
Takt
T
Steuer-
werk
SW
8 B_{23}-B_{16}
Z
-Z
24
C_2-C_0
3
interner
Datenbus
24
ALU
X
-X
24
Y
-Y
24
S
D_r
D_w
SAR
SDR
20
24
20
A_{19}-A_0
24
D_{23}-D_0
R
W

MIMA Program Beispiel

- **(C Code:)**

  */*int x,y = 1,2*

  &x = 0x0000

  &y = 0x0002/*

  int z = x + y;

- **MIMA Assembler:**

  LDV 0x0000;

  ADD 0x0002;

  STV 0x0004;

  TEST 0x0004;

- **Mikroprogramm**
  - *Lese-Phase*
  - *Dekodierphase*
  - *Ausführungsphase*

Mikroprogramm LDV Lese-Phase

Register-Tranfer Schreibweise:

1. Takt: IAR -> SAR; IAR -> X; R = 1;
2. Takt: Eins -> Y; R = 1;
3. Takt: ALU auf Addieren; R = 1
4. Takt: Z -> IAR
5. Takt: SDR -> IR

Ober binär Codiert:

```
0010 0001 0000 1000 1000 0000 0001
0001 0100 0000 0000 1000 0000 0010
0000 0000 0000 0001 1000 0000 0011
0000 1010 0000 0000 0000 0000 0100
0000 0000 1001 0000 0000 0000 0101
```

Mikroprogramm LDV Decodierphase

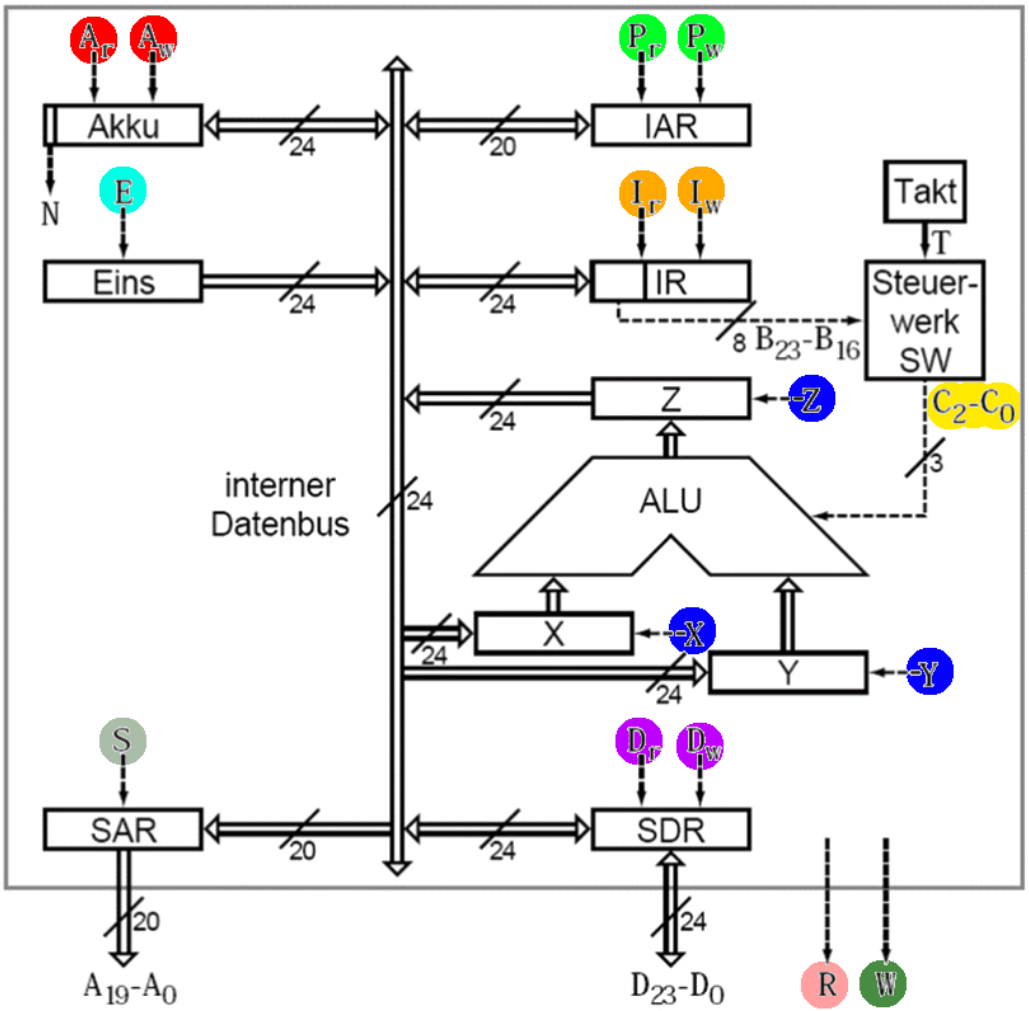
6 Dekodiere Befehl im IR

-> Nutzen eines Demultiplexers zur Auswahl des nächsten Mikroprogramms

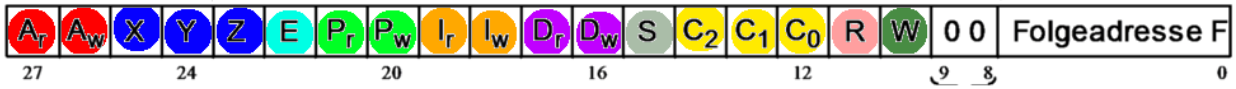
Mikroprogramm LDV
Ausführungsphase

Karlsruher Institut für Technologie

7. Takt: IR -> SAR; R = 1
8. Takt: R = 1
9. Takt: R = 1
10. Takt: SDR -> Akku



Mikrobefehlsformat



Aufgabe 6.1 (Klausur SS_16)

1. Takt:	IAR $\rightarrow$ SAR; IAR $\rightarrow$ X; R = 1	} Lese-Phase
2. Takt:	Eins $\rightarrow$ Y; R = 1	
3. Takt:	ALU auf Addieren; R = 1	
4. Takt:	Z $\rightarrow$ IAR	
5. Takt:	SDR $\rightarrow$ IR	

Lösung 6.1 (Klausur SS_16)

Takt	Adresse	Befehl in hexadezimaler Schreibweise
1. Takt	0x00	2 1 0 8 8 0 1 $(X = P_w = S = 1; R = 1)$
2. Takt	0x01	1 4 0 0 8 0 2 $(Y = E = 1; R = 1)$
3. Takt	0x02	0 0 0 1 8 0 3 $(C_2\text{-}C_0 = 001; R = 1)$
4. Takt	0x03	0 A 0 0 0 0 4 $(Z = P_r = 1)$
5. Takt	0x04	0 0 9 0 0 0 5 $(I_r = D_w = 1)$

Aufgabe 6.2 (Klausur SS_16)

Aufgabe

Geben Sie die Mikroprogramme für die Ausführungsphasen der folgenden Maschinenbefehle an (jeweils ab dem 7. Takt, also nach der Lese-Phase und der DekodierPhase):

LDV, STV

Lösung 6.2 (Klausur SS_16)

LDV		STV	
7. Takt:	IR → SAR; R = 1	7. Takt:	Akku → SDR
8. Takt:	R = 1	8. Takt:	IR → SAR; W = 1
9. Takt:	R = 1	9. Takt:	W = 1
10. Takt:	SDR → Akku	10. Takt:	W = 1

Aufgabe 6.2 (Klausur SS_16)

Aufgabe

Geben Sie die Mikroprogramme für die Ausführungsphasen der folgenden Maschinenbefehle an (jeweils ab dem 7. Takt, also nach der Lese-Phase und der DekodierPhase):

EQL, JMP

Lösung 6.2 (Klausur SS_16)

EQL		JMP	
7. Takt:	IR $\rightarrow$ SAR; R = 1	7. Takt:	IR $\rightarrow$ IAR
8. Takt:	Akku $\rightarrow$ X; R = 1		
9. Takt:	R = 1		
10. Takt:	SDR $\rightarrow$ Y		
11. Takt:	ALU auf Vergleich		
12. Takt:	Z $\rightarrow$ Akku		

Roadmap

Aufgabe 7.1 (Klausur SS_17)

Aufgabe

Schreiben Sie die folgende C-Kontrollstruktur in MIPS-Assembler um.

```
for (i = 100; i > 0; i--)
    j = j * i;
```

Die Variablen i und j stehen in den Registern $a0 und $a1. Verwenden Sie das Register $v0 zur Speicherung temporärer Variablen.

Aufgabe 7.1 (Klausur SS_17)

Lösung

```
        addi $a0, $zero, 100      # i = 100;
loop:   slt $v0, $zero, $a0       # if 0 < i, dann $v0 = 1
        beq $v0, $zero, label     # if $v0 = 0 (d.h., i <= 0)

        # dann Ende der for-Schleife
        mul $a1, $a1, $a0         # j = j * i;
        subi  $a0, $a0, 1         # i--;
        b loop                    # gehe zur Marke loop
label:
```

Aufgabe 7.2 (Klausur SS_17)

Aufgabe

Das folgende Programmstück soll die Summe der Elemente eines Arrays aus 32- Bit Integer-Zahlen in Zweierkomplement-Form berechnen. Das Register $a0 sei mit der Adresse des Arrays initialisiert; das Register $a1 sei mit der Anzahl der ArrayElemente initialisiert. Alle anderen Register seien nicht initialisiert.

```
ADD:  li  $t0, 0($a0)
      add $v0, $v0, $t0
      addi $a0, $a0, 4
      addi $a1, $a1, -1
      bgez $a0, ADD
```

Leider haben sich bei der Implementierung einige Fehler eingeschlichen. Finden Sie diese Fehler und korrigieren Sie das Programm, so dass es korrekt arbeitet.

Aufgabe 7.2 (Klausur SS_17)

Lösung

```
        add $v0, $zero, $zero # summe = 0
ADD:    lw $t0, 0($a0)        # Naechstes Element lesen
        add $v0, $v0, $t0     # Addieren
        addi $a0, $a0, 4      # Zeiger auf das naechste Element
        addi $a1, $a1, -1     # Zaehler dekrementieren
        bgtz $a1, ADD         # Abbruchbedingung
```

Aufgabe 7.3 (Klausur SS_17)

Aufgabe

Geben Sie für das folgende MIPS-Programmstück den Inhalt des Zielregisters in hexadezimaler Schreibweise nach der Ausführung des jeweiligen Befehls an.

```
subi    $s1, $zero, 0x2
srl     $s2, $s1, 4
slti    $s3, $s2, 100
lui     $s4, 0x40
xor     $s5, $s1, $s4
```

Lösung 7.3 (Klausur SS_17)

Befehl	Zielregister = (z. B. $s6 = 0x0000 F00A)
`subi $s1, $zero, 0x2`	`$s1 = 0xFFFF FFFE`
`srl $s2, $s1, 4`	`$s2 = 0x0FFF FFFF`
`slti $s3, $s2, 100`	`$s3 = 0x0000 0000`
`lui $s4, 0x40`	`$s4 = 0x0040 0000`
`xor $s5, $s1, $s4`	`$s5 = 0xFFBF FFFE`

Aufgabe 7.4 (Klausur SS_17)

Aufgabe

Gegeben sei das folgende MIPS-Programmstück:

```
.data
vec:    .word 12, 13, 17, 19, 23, 29, 31, 37, 41, 43

.text
main:   lw   $t1, vec
        lw   $t2, vec+0x18
        lw   $t3, vec($t1)
        lw   $t4, vec+0x14($t1)
```

- Geben Sie die Inhalte der Register $t1, $t2, $t3 und $t4 in hexadezimaler Schreibweise nach der Ausführung des obigen Programmcodes an.
- Geben Sie MIPS-Code an, mit dem man die Adresse von vec im Register $s0 speichert.

Lösung 7.4 (Klausur SS_17)

(a) Registerinhalte:

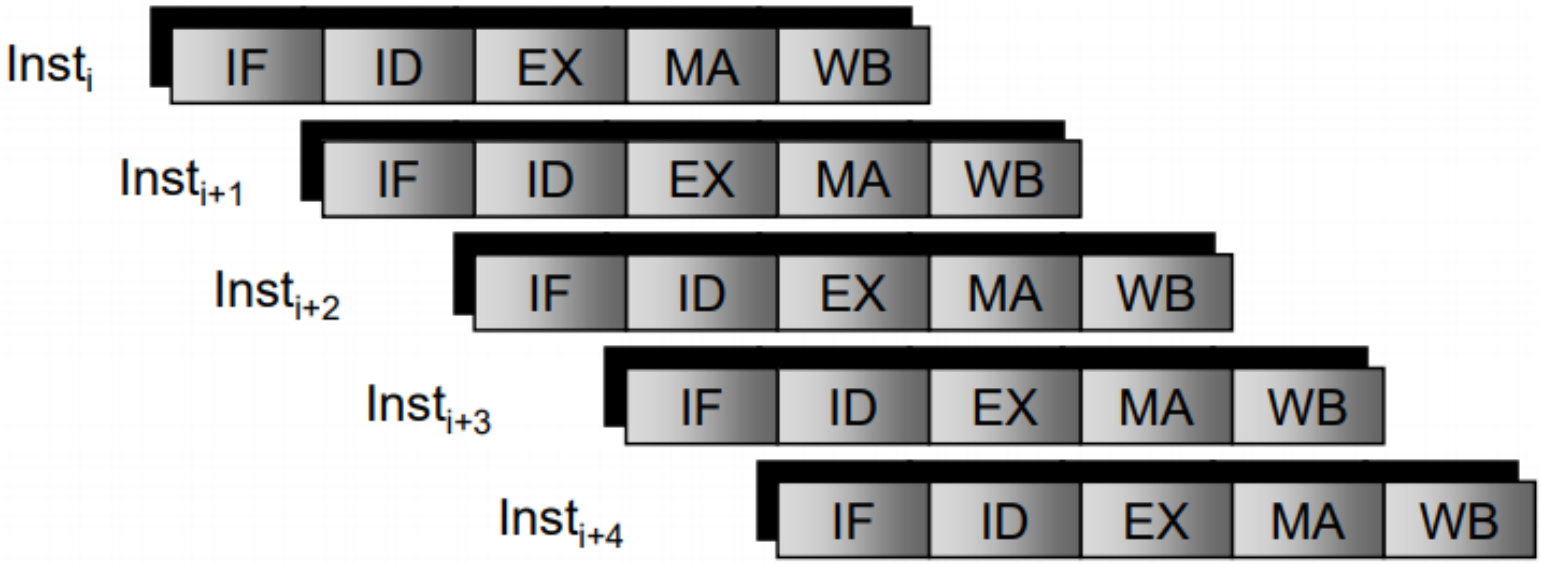
Register	Inhalt
`$t1`	`$t1 = 0x0000 000C`
`$t2`	`$t2 = 0x0000 001F`
`$t3`	`$t3 = 0x0000 0013`
`$t4`	`$t4 = 0x0000 0029`

(b) MIPS-Code zur Speicherung der Adresse von `vec` im Register `$s0`:

```
la $s0, vec
```

Roadmap

Pipeline



Logische Phasen:
IF: Befehl holen
ID: Befehl dekodieren / Operanden bereitstellen
EX: Befehl ausführen
MA: Speicherzugriff
WB: Zurückschreiben

Pipeline-Konflikte

- Datenkonflikte
  - echte Datenabhängigkeit
  - Gegenabhängigkeit
  - Ausgabeabhängigkeit
- Steuerkonflikte
- Strukturkonflikte

Beispiel

- **Beispiel:**

```
S1:     add     r1,r2,2         #       r1 := r2 + 2
S2:     add     r4,r1,r3        #       r4 := r1 + r3
S3:     mul     r3,r5,3         #       r3 := r5 * 3
S4:     mul     r3,r6,3         #       r3 := r6 * 3
```

Beispiel

- **Beispiel:**

```
S1:     add     r1,r2,2         #       r1 := r2 + 2
S2:     add     r4,r1,r3        #       r4 := r1 + r3
S3:     mul     r3,r5,3         #       r3 := r5 * 3
S4:     mul     r3,r6,3         #       r3 := r6 * 3
```

Echte Datenabhängigkeit δ^t

Beispiel

- **Beispiel:**

```
S1:     add     r1,r2,2         #       r1 := r2 + 2
S2:     add     r4,r1,r3        #       r4 := r1 + r3
S3:     mul     r3,r5,3         #       r3 := r5 * 3
S4:     mul     r3,r6,3         #       r3 := r6 * 3
```

Gegenabhängigkeit δ^a

Beispiel

- **Beispiel:**

```
S1:     add     r1,r2,2         #       r1 := r2 + 2
S2:     add     r4,r1,r3        #       r4 := r1 + r3
S3:     mul     r3 r5,3         #       r3 := r5 * 3
S4:     mul     r3 r6,3         #       r3 := r6 * 3
```

Ausgabeabhängigkeit δ^o

Aufgabe 7.1 (Klausur SS_18)

Aufgabe

Das folgende Programmstück soll in der DLX-Pipeline abgearbeitet werden. Geben Sie alle Datenabhängigkeiten an.

```
S1:   addi    $t1, $t2, 7
S2:   sub     $t4, $t3, $t1
S3:   muli    $t3, $t5, 23
S4:   addi    $t3, $t3, 13
S5:   add     $t2, $t1, $t5
```

Aufgabe 7.1 (Klausur SS_18)

Lösung

- True Dependence (δ^t)

  S_1 -> S_2 | S_3 -> S_4 | S_1 -> S_5

Aufgabe 7.1 (Klausur SS_18)

Lösung

- True Dependence (δ^t)

  S_1 -> S_2 | S_3 -> S_4 | S_1 -> S_5

- Anti Dependence (δ^a)

  S_2 -> S_3 | S_2 -> S_4 | S_1 -> S_5

Aufgabe 7.1 (Klausur SS_18)

Lösung

- True Dependence (δ^t)
  S_1 -> S_2 | S_3 -> S_4 | S_1 -> S_5
- Anti Dependence (δ^a)
  S_2 -> S_3 | S_2 -> S_4 | S_1 -> S_5
- Output Dependence (δ^o)
  S_3 -> S_4

Aufgabe 7.2 (Klausur SS_18)

Aufgabe

Zu Beginn des Programmstücks seien die Register folgendermaßen belegt:

$t1	$t2	$t3	$t4	$t5
3	5	7	9	2

Der Prozessor verfügt über keine Möglichkeiten Pipelinekonflikte zu beheben. Tragen Sie in die Tabelle auf dem Lösungsblatt den Zustand von Pipeline und Registern ein. Und zwar für alle Takte (bis der letzte Befehl zurückgeschrieben hat) jeweils am Ende des Taktes. Wie viele Takte werden benötigt, um das Programm abzuarbeiten?

Aufgabe 7.2 (Klausur SS_18)

Aufgabe

```
S1:   addi    $t1, $t2, 7
S2:   sub     $t4, $t3, $t1
S3:   muli    $t3, $t5, 23
S4:   addi    $t3, $t3, 13
S5:   add     $t2, $t1, $t5
```

Aufgabe 7.2 (Klausur SS_18)

Takt	IF	ID/RF	EX	MEM	WB	$t1	$t2	$t3	$t4	$t5
1	S1	—	—	—	—	3	5	7	9	2
2	S2	S1	—	—	—	3	5	7	9	2
3	S3	S2	S1	—	—	3	5	7	9	2
4	S4	S3	S2	S1	—	3	5	7	9	2
5	S5	S4	S3	S2	S1	12	5	7	9	2
6	—	S5	S4	S3	S2	12	5	7	4	2
7	—	—	S5	S4	S3	12	5	46	4	2
8	—	—	—	S5	S4	12	5	20	4	2
9	—	—	—	—	S5	12	14	20	4	2

Aufgabe 7.3 (Klausur SS_18)

Aufgabe

Die einzige Methode, die Pipelinekonflikte bei diesem Prozessor zu beheben, sei das Einfügen von NOP-Befehlen (No Operation) in den Befehlsstrom. Fügen Sie so wenig wie möglich NOP-Befehle in das Programmstück ein, so dass es zu keinen Konflikten mehr kommt. Aber ohne die bereits vorhandenen Befehle oder ihre Reihenfolge zu verändern. Geben Sie das modifizierte Programmstück an. Wie viele Takte werden nun benötigt?

```
S1:   addi    $t1, $t2, 7
S2:   sub     $t4, $t3, $t1
S3:   muli    $t3, $t5, 23
S4:   addi    $t3, $t3, 13
S5:   add     $t2, $t1, $t5
```

Aufgabe 7.3 (Klausur SS_18)

Lösung

```
S1:   addi    $t1, $t2, 7
      nop
      nop
S2:   sub     $t4, $t3, $t1
S3:   muli    $t3, $t5, 23
      nop
      nop
S4:   addi    $t3, $t3, 13
S5:   add     $t2, $t1, $t5
```

Aufgabe 7.4 (Klausur SS_18)

Aufgabe

Warum stellen bedingte Sprünge ein Problem für die Pipelineimplementierung dar? Geben Sie zwei Möglichkeiten zur Behandlung dieses Problems an

Aufgabe 7.4 (Klausur SS_18)

Aufgabe

Warum stellen bedingte Sprünge ein Problem für die Pipelineimplementierung dar? Geben Sie zwei Möglichkeiten zur Behandlung dieses Problems an

Lösung

Problem: Erst am Ende der 4. Stufe wird entscheiden, ob gesprungen wird oder nicht. Die folgenden drei Befehle sind schon in der Pipeline und müssen im Falle eines Sprungs gelöscht werden.

Aufgabe 7.4 (Klausur SS_18)

Lösung

Behandlungsmöglichkeiten:

- Verzögerte Sprungtechnik (*delayed branch technique*): z.B. drei Verzögerungszeitschlitze (*delay slots*) mit Leerbefehlen (**NOP**) nach jedem Sprungbefehl.
- Befehlsumordnung: Befehle, die in der logischen Programmreihenfolge vor dem Sprungbefehl liegen, in die Verzögerungszeitschlitze verschieben.
- Pipeline-Leerlauf (ineffizient): Die Hardware erkennt die Verzweigungsbefehle in der ID-Phase und lädt keine weiteren Befehle in die Pipeline, bis die Zieladresse berechnet und im Fall bedingter Sprungbefehle die Sprungentscheidung getroffen ist.
- Sprungvorhersage: Spekulation über Sprungentscheidung und/oder Sprungziel. –> möglichst wenig Pipeline-Leerlauf/Delay Slots.

Was ihr jetzt kennen und können solltet...

- Nichts neues...
- Ihr weisst jetzt wie eine Klausur aufgebaut ist ;)

Das wars