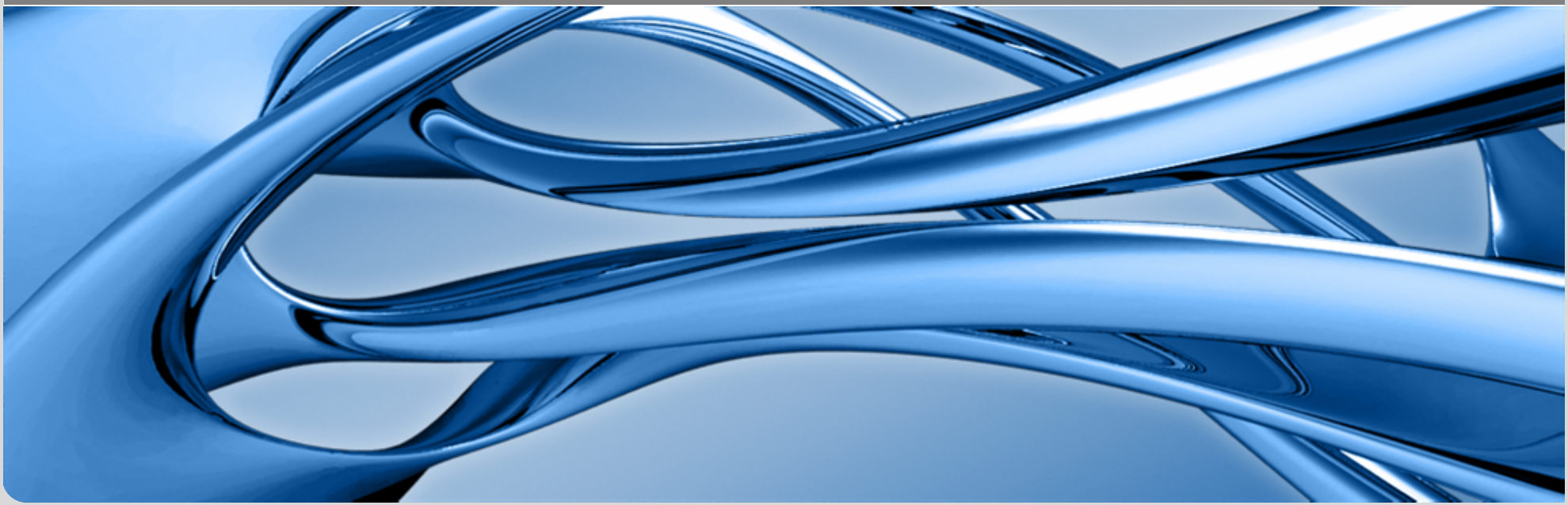
Karlsruher Institut für Technologie

3. Tutorium Rechnerorganisation

Tutorium 7| WS19/20

Grégoire Mercier | 17. November 2019

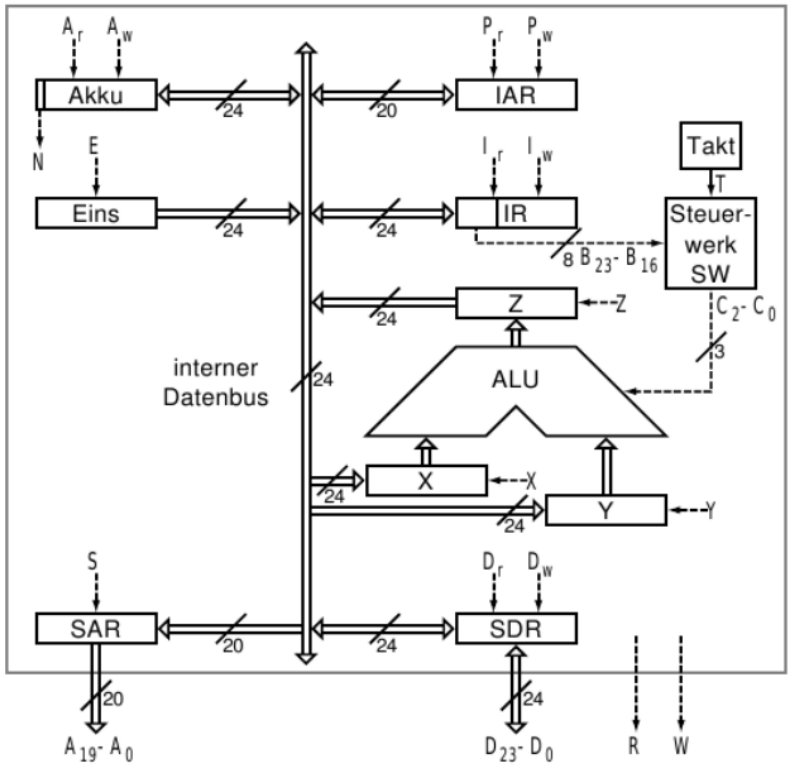
INSTITUT FÜR TECHNISCHE INFORMATIK



Gliederung

Die Architektur der MIMA

Architektur der MIMA



Register

Akku: Akkumulator
X: 1. ALU Operand
Y: 2. ALU Operand
Z: ALU Ergebnis
Eins: Konstante 1
IAR: Instruktionsadreßregister
IR: Instruktionsregister
SAR: Speicheradreßregister
SDR: Speicherdatenregister

Steuersignale vom SW

– für den internen Datenbus

A_r: Akku liest
A_w: Akku schreibt
X: X-Register liest
Y: Y-Register liest
Z: Z-Register schreibt
E: Eins-Register schreibt
P_r: IAR liest
P_w: IAR schreibt
I_r: IR liest
I_w: IR schreibt
D_r: SDR liest
D_w: SDR schreibt
S: SAR liest

– für die ALU

C_2-C_0: Operation auswählen

– für den Speicher

R: Leseanforderung
W: Schreibanforderung

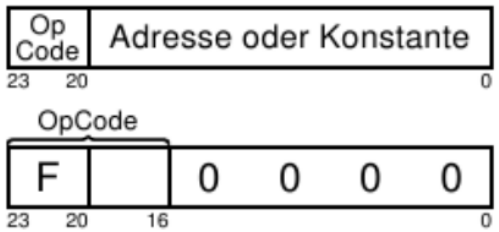
Meldesignale zum SW

T: Takteingang
N: Vorzeichen des Akku
B_{23}-B_{16}: OpCode-Feld im IR

Befehle der MIMA

OpCode	*Mnemonik*	*Beschreibung*
0	LDC c	c -> Akku
1	LDV a	<a> -> Akku
2	STV a	Akku -> <a>
3	ADD a	Akku + <a> -> Akku
4	AND a	Akku AND <a> -> Akku
5	OR a	Akku OR <a> -> Akku
6	XOR a	Akku XOR <a> -> Akku
7	EQL a	falls Akku = <a>:-1 -> Akku
		sonst : 0 -> Akku
8	JMP a	a -> IAR
9	JMN a	falls Akku < 0 : a -> IAR
F0	HALT	stoppt die MIMA
F1	NOT	bilde Eins-Komplement von Akku -> Akku
F2	RAR	rotiere Akku eins nach rechts -> Akku

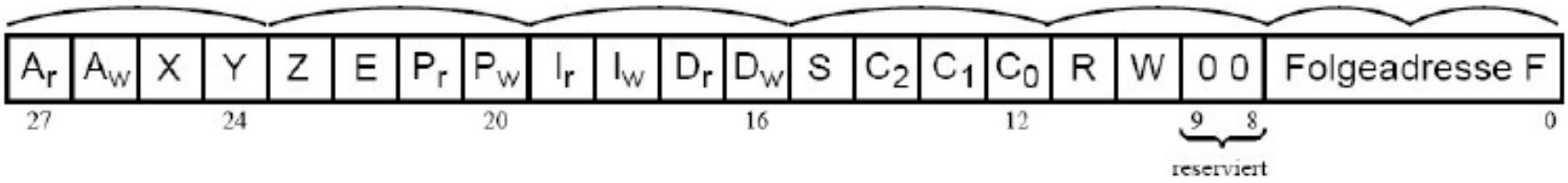
Befehlsformate



ALU-Operationen der MIMA

$C_2 C_1 C_0$	*ALU Operation*
0 0 0	tue nichts (d.h. Z -> Z)
0 0 1	X + Y -> Z
0 1 0	rotiere X nach rechts -> Z
0 1 1	X AND Y -> Z
1 0 0	X OR Y -> Z
1 0 1	X XOR Y -> Z
1 1 0	Eins-Komplement von X -> Z
1 1 1	falls X = Y, -1 -> Z, sonst 0 -> Z

Mikrobefehlsformat der MIMA



Übung 1

- MIMA wird um den Befehl JMS (*jump subroutine*) erweitert.
- Erhält OP-Code C
- Dieser speichert die Folgeadresse in <a> und springt auf die Folgeadresse von a

JMS a $IAR + 1 \rightarrow < a >; a + 1 \rightarrow IAR$

- Wie lautet das Programm für die Ausführungsphase?

Übung 1

- MIMA wird außerdem um den Befehl JIND (*jump indirect*) erweitert
- Erhält OP-Code D
- Dieser springt zur Adresse, die an Speicherstelle a gespeichert ist
- Da Speicherstellen 24 Bits lang sind, durch den Befehl aber nur 20 Bits geladen werden können, werden diese nicht berücksichtigt

JIND a $\qquad\qquad < a > \rightarrow IAR$

- Wie lautet das Programm für die Ausführungsphase

Lösung

JMS:

7. Takt:	$\text{IAR} \rightarrow \text{SDR};$
8. Takt:	$\text{IR} \rightarrow \text{SAR}; \text{IR} \rightarrow \text{X}; \text{W} = 1$
9. Takt:	$\text{EINS} \rightarrow \text{Y}; \text{W} = 1$
10. Takt:	$\text{ALU auf ADD}; \text{W} = 1$
11. Takt:	$\text{Z} \rightarrow \text{IAR}$

Lösung

JIND:

7. Takt:	$IR \rightarrow SAR;$	$R = 1$
8. Takt:	$R = 1$	
9. Takt:	$R = 1$	
10. Takt:	$SDR \rightarrow IAR$	

Übung 2

- Schreibt ein MIMA-Programm, das zu einem beliebigen Tag (TAG, MONAT) im Jahr 2020 den entsprechenden Wochentag berechnet
- Dazu werden Wochentage wie folgt definiert:

  Sonntag = 0, **Montag = 1**, ... , **Samstag = 6**
- Gegeben sind folgende Speicherstellen:
  - 0x00001 Erster Monatskalendereintrag
  - 0x00020 TAG
  - 0x00021 MONAT
  - 0x00030 Ausgabe Wochentag
  - 0x00100 Beginn des Programms

Übung 2

Orientiert euch an folgendem C-Code:

```
main() {
        int ersterTag[] = {-1,3,6,0,3,5,1,3,6,2,4,0,2};
        int TAG = 3;
        int MONAT = 6;
        int wochentag = tag + ersterTag[MONAT] - 1;

        wochentag %= 7;

        printf("Wochentag %i\n", wochentag)
}
```

Kostanten speichern wir an den Adressen 0x00001 bis 0x0000C (-1 ist hier nur ein Platzhalter, damit auch in C der Monat mit 1 anfangen kann)

```
0x00001 Array:   DS 3
...
0x0000C          DS 2
```

```
int wochentag = tag + ersterTag[MONAT] - 1;
```

-1 als Konstante in 0x00023 (Label MINUS1)

```
0x00100:        START:  LDV     MONAT
0x00101:                ADD     BEFEHL
0x00102:                STV     BEFEHL
0x00103:        BEFEHL: LDV     0
0x00104:                ADD     TAG
0x00105:                ADD     MINUS1
```

```
wochentag %-= 7;
```

Konstanten -7 und +7 in entsprechenden Labels und dann noch an der gewünschten Stelle speichern (Dafür wurde 0x00030 RESULT: DS 0x0 reserviert)

```
0x00106:        LOOP:   ADD     MINUS7
0x00107:                JMN     EXIT
0x00108:                JMP     LOOP
0x00109:        EXIT:   ADD     PLUS7
0x0010A:                STV     RESULT
0x0010B:        ENDE:   HALT
```

Danke für eure Aufmerksamkeit!