

Übersicht

(1. Allgemeine Grundlagen)

2. Spezielle Grundlagen

3. Vorversuche

4. Versuche

Übersicht

(1. Allgemeine Grundlagen)

2. Spezielle Grundlagen

3. Vorversuche

4. Versuche

Übersicht

(1. Allgemeine Grundlagen)

2. Spezielle Grundlagen

3. Vorversuche

4. Versuche

DEP-CPU: Dynamische Eigenschaften

DEP-CPU: Dynamische Eigenschaften

- **Vorgabe: Abarbeitung von Maschinenbefehlen**

DEP-CPU: Dynamische Eigenschaften

- Vorgabe: Abarbeitung von Maschinenbefehlen
- **Notwendig:**
 - 1. Befehlsunabhängige Schritte**

DEP-CPU: Dynamische Eigenschaften

- Vorgabe: Abarbeitung von Maschinenbefehlen
- Notwendig:
 1. Befehlsunabhängige Schritte
 - 2. Befehlsabhängige Schritte**

DEP-CPU: Dynamische Eigenschaften

- Vorgabe: Abarbeitung von Maschinenbefehlen
 - Notwendig:
 1. Befehlsunabhängige Schritte
 2. Befehlsabhängige Schritte
- **3 Zyklen für jeden Maschinenbefehl:**
1. Befehl holen (1 Takt)

DEP-CPU: Dynamische Eigenschaften

- Vorgabe: Abarbeitung von Maschinenbefehlen
 - Notwendig:
 1. Befehlsunabhängige Schritte
 2. Befehlsabhängige Schritte
- 3 Zyklen für jeden Maschinenbefehl:
1. Befehl holen (1 Takt)
 - 2. Operand holen (1 Takt)**

DEP-CPU: Dynamische Eigenschaften

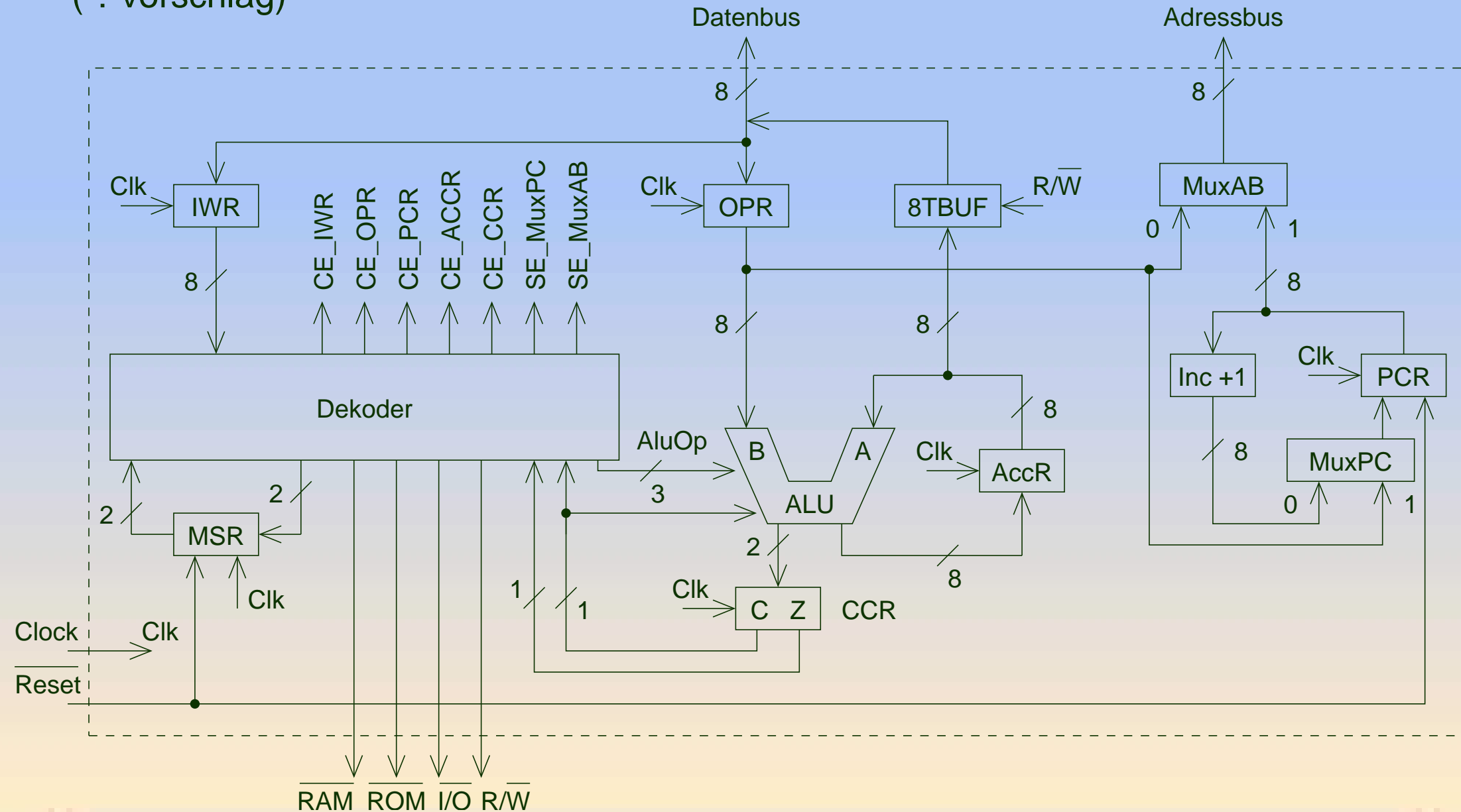
- Vorgabe: Abarbeitung von Maschinenbefehlen
 - Notwendig:
 1. Befehlsunabhängige Schritte
 2. Befehlsabhängige Schritte
- 3 Zyklen für jeden Maschinenbefehl:
1. Befehl holen (1 Takt)
 2. Operand holen (1 Takt)
 - 3. Befehl ausführen (1-2 Takte)**

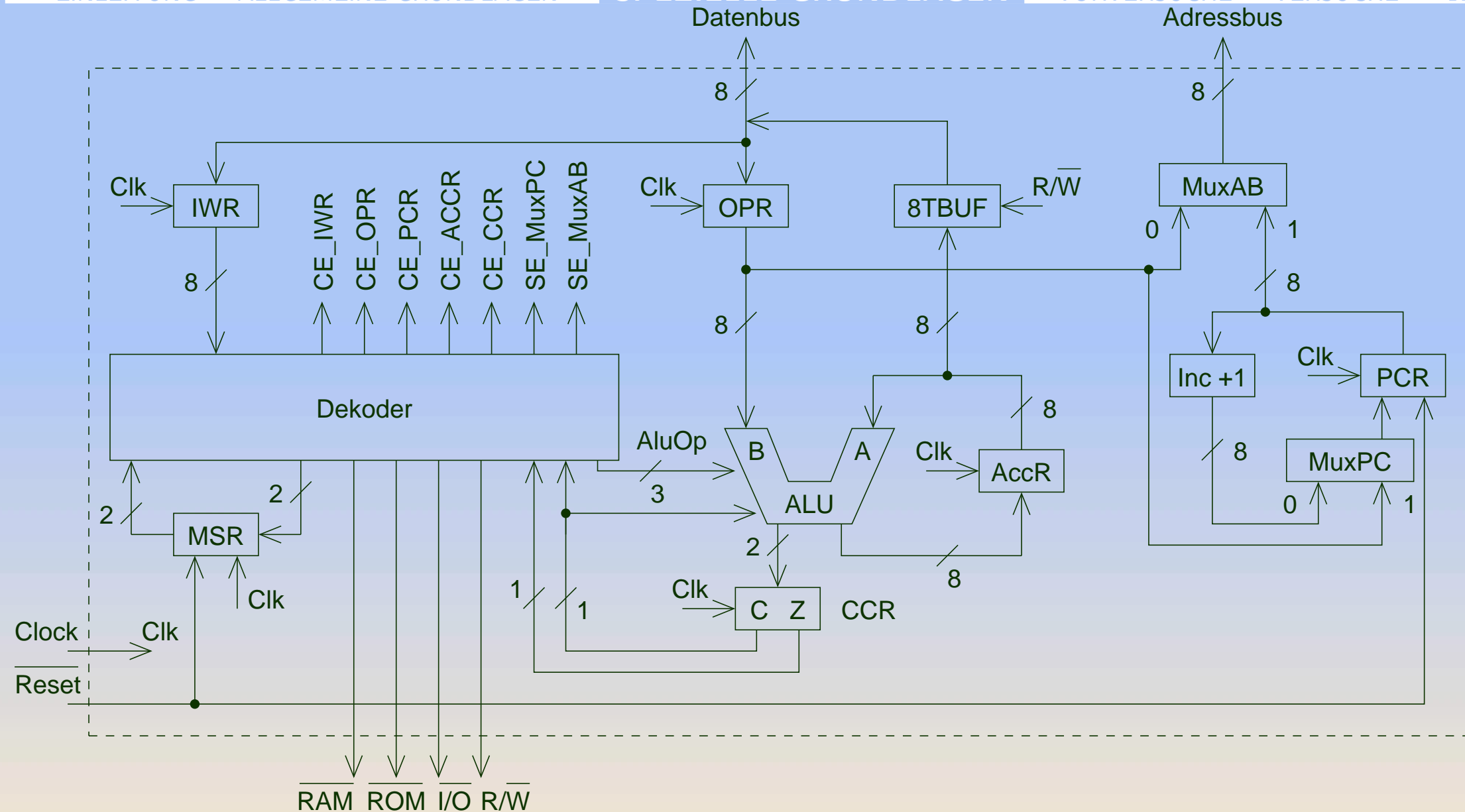
DEP: CPU-Architektur*

(*: Vorschlag)

DEP: CPU-Architektur*

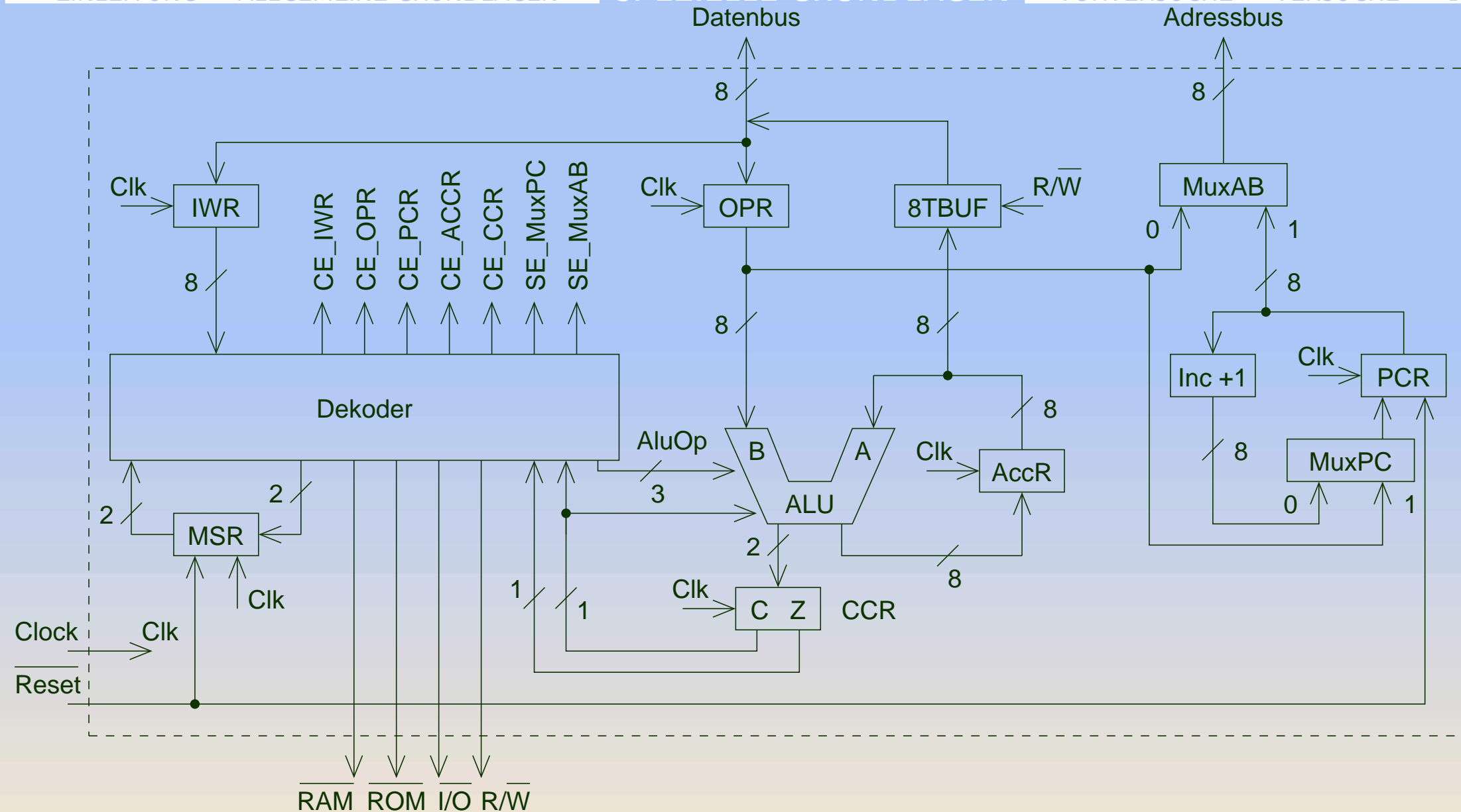
(*: Vorschlag)



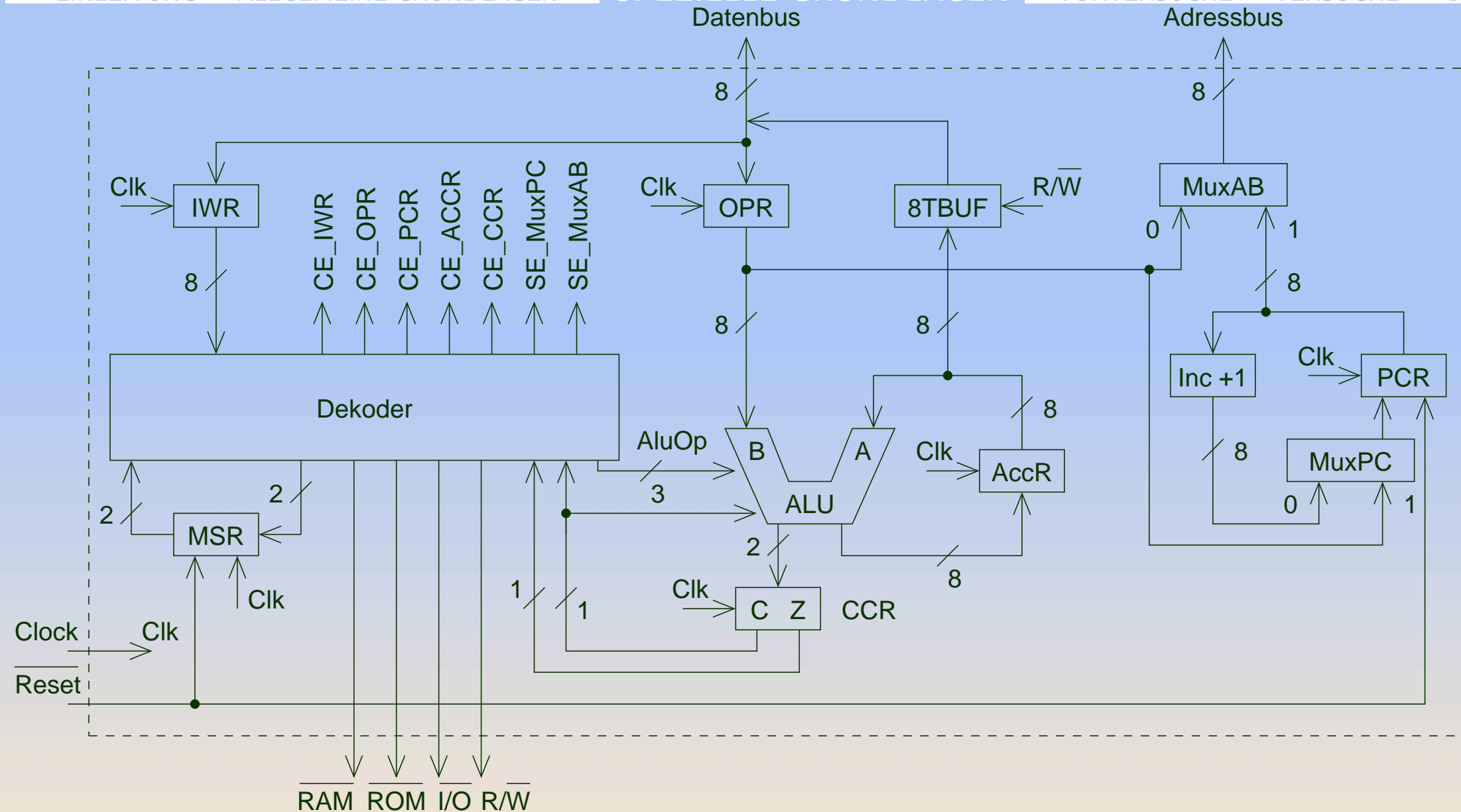


RAM ROM I/O R/W

Instruction Word	CCR:Z,C		MSR out	MSR in	IWR	OPR	MuxAB	MuxPC	AccR	CCR	PCR	$\overline{\text{RAM}}$	$\overline{\text{ROM}}$	$\overline{\text{I/O}}$	$\overline{\text{R/W}}$	AluOp
—	—	—	0	1	1	0	1	0	0	0	1	1	0	1	1	—



Instruction Word	CCR:Z,C		MSR out	MSR in	IWR	OPR	MuxAB	MuxPC	AccR	CCR	PCR	\overline{RAM}	\overline{ROM}	$\overline{I/O}$	$\overline{R/W}$	\overline{AluOp}
–	–	–	0	1	1	0	1	0	0	0	1	1	0	1	1	–
–	–	–	1	2	0	1	1	0	0	0	1	1	0	1	1	–



Instruction Word	CCR:Z,C	MSR out	MSR in	IWR	OPR	MuxAB	MuxPC	AccR	CCR	PCR	\overline{RAM}	\overline{ROM}	\overline{IO}	$\overline{R/W}$	AluOp
STA 0x80	-	-	2	0	0	0	-	0	0	0	0	1	1	0	-

DEP: Instruktionssatz

DEP: Instruktionssatz

Gruppierung der Befehle nach Funktion:

DEP: Instruktionssatz

Gruppierung der Befehle nach Funktion:

- Transferbefehle: LDA, LDAI, STA, STAX, IN, OUT

DEP: Instruktionssatz

Gruppierung der Befehle nach Funktion:

- Transferbefehle: LDA, LDAI, STA, STAX, IN, OUT
- Arithmetikbefehle: ADD, OR, SBC, XOR, ADC, AND, SETF,
ADDI, ORI, SBCI, XORI, ADCI, ANDI, SETFI

DEP: Instruktionssatz

Gruppierung der Befehle nach Funktion:

- Transferbefehle: LDA, LDAI, STA, STAX, IN, OUT
- Arithmetikbefehle: ADD, OR, SBC, XOR, ADC, AND, SETF,
ADDI, ORI, SBCI, XORI, ADCI, ANDI, SETFI
- Sprungbefehle: BCS, BEQ, JMP, JMPI

DEP: Instruktionssatz

Gruppierung der Befehle nach Adressierungsart:

DEP: Instruktionssatz

Gruppierung der Befehle nach Adressierungsart:

- Unmittelbar (Immediate): LDAI, ADDI, ORI, SBCI, XORI, ADCI, ANDI, SETFI, JMPI, BCS, BEQ, (STA, OUT)

DEP: Instruktionssatz

Gruppierung der Befehle nach Adressierungsart:

- Unmittelbar (Immediate): LDAI, ADDI, ORI, SBCI, XORI, ADCI, ANDI, SETFI, JMPI, BCS, BEQ, (STA, OUT)
- Direkt: (STA, OUT,) LDA, IN, ADD, OR, SBC, XOR, ADC, AND, SETF, JMP

DEP: Instruktionssatz

Gruppierung der Befehle nach Adressierungsart:

- Unmittelbar (Immediate): LDAI, ADDI, ORI, SBCI, XORI, ADCI, ANDI, SETFI, JMPI, BCS, BEQ, (STA, OUT)
- Direkt: (STA, OUT,) LDA, IN, ADD, OR, SBC, XOR, ADC, AND, SETF, JMP
- Indirekt: STAX

DEP: Instruktionssatz

Gruppierung der Befehle nach Adressierungsart:

- Unmittelbar (Immediate): LDAI, ADDI, ORI, SBCI, XORI, ADCI, ANDI, SETFI, JMPI, BCS, BEQ, (STA, OUT)
- Direkt: (STA, OUT,) LDA, IN, ADD, OR, SBC, XOR, ADC, AND, SETF, JMP
- Indirekt: STAX
- Keine Register-basierten Adressierungsarten (“registered indirect”).

DEP: Instruktionssatz

Sedezimalcode	Befehlskürzel	Funktion	beeinflusste Flags
Speichertransferbefehle			
80 xx	STA xx	Speichere Inhalt von A an Adresse xx	
A0 zz	STAX zz	Speichere Inhalt von A an Adresse xx, die an Adresse zz steht	
40 xx	LDA xx	Lade A mit Inhalt an Adresse xx	Z
48 yy	LDAI yy	Lade A mit Konstante yy	Z
Sprungbefehle			
28 dd	BCS dd	Wenn Carry-Flag gesetzt: PCR:=dd	
18 dd	BEQ dd	Wenn Zero-Flag gesetzt: PCR:=dd	
30 xx	JMP xx	Setze PCR auf den Inhalt an Adresse xx	
38 dd	JMPI dd	Setze PCR auf dd	
Ein-/Ausgabe-Befehle			
50 xx	IN xx	Lade A mit Inhalt an Portadresse xx	Z
90 xx	OUT xx	Schreibe Inhalt von A an Portadresse xx	
Rechenwerksbefehle (Speicher)			
41 xx	ADD xx	Addiere A zu Inhalt von xx	C und Z
42 xx	OR xx	Bitweises Oder von A und [xx]	Z
43 xx	SBC xx	$A := A - [xx] - C$	C und Z
44 xx	XOR xx	Bitweises ExOder von A und [xx]	Z
45 xx	ADC xx	$A := A + [xx] + C$	C und Z
46 xx	AND xx	Bitweises Und von A und [xx]	Z
47 xx	SETF xx	Setze C und Z entsprechend Bit 6 und 7 von [xx] und behalte A unverändert bei	C und Z
Rechenwerksbefehle (Immediate, also konstanter Operand)			
49 yy	ADDI yy	Addiere A zu yy	C und Z
4A yy	ORI yy	Bitweises Oder von A und yy	Z
4B yy	SBCI yy	$A := A - yy - C$	C und Z
4C yy	XORI yy	Bitweises ExOder von A und yy	Z
4D yy	ADCI yy	$A := A + yy + C$	C und Z
4E yy	ANDI yy	Bitweises Und von A und yy	Z
4F yy	SETFI yy	Setze C und Z entsprechend Bit 6 und 7 von yy und behalte A unverändert bei	C und Z

DEP: Instruktionssatz

Sedezimalcode	Befehlskürzel	Funktion	beeinflusste Flags	
Speichertransferbefehle				
80	xx	STA xx	Speichere Inhalt von A an Adresse xx	
A0	zz	STAX zz	Speichere Inhalt von A an Adresse xx, die an Adresse zz steht	
40	xx	LDA xx	Lade A mit Inhalt an Adresse xx	Z
48	yy	LDAI yy	Lade A mit Konstante yy	Z
Ein-/Ausgabe-Befehle				
50	xx	IN xx	Lade A mit Inhalt an Portadresse xx	Z
90	xx	OUT xx	Schreibe Inhalt von A an Portadresse xx	

DEP: Instruktionssatz

beeinflusste
Flags

Sedezimalcode Befehlskürzel Funktion

Sprungbefehle

28	dd	BCS	dd	Wenn Carry-Flag gesetzt: PCR:=dd
18	dd	BEQ	dd	Wenn Zero-Flag gesetzt: PCR:=dd
30	xx	JMP	xx	Setze PCR auf den Inhalt an Adresse xx
38	dd	JMPI	dd	Setze PCR auf dd

DEP: Instruktionssatz

Sedezimalcode	Befehlskürzel	Funktion	beeinflusste Flags
Rechenwerksbefehle (Speicher)			
41 xx	ADD xx	Addiere A zu Inhalt von xx	C und Z
42 xx	OR xx	Bitweises Oder von A und [xx]	Z
43 xx	SBC xx	$A := A - [xx] - C$	C und Z
44 xx	XOR xx	Bitweises ExOder von A und [xx]	Z
45 xx	ADC xx	$A := A + [xx] + C$	C und Z
46 xx	AND xx	Bitweises Und von A und [xx]	Z
47 xx	SETF xx	Setze C und Z entsprechend Bit 6/7 von [xx] und behalte A unverändert bei	C und Z
Rechenwerksbefehle (Immediate, also konstanter Operand)			
49 yy	ADDI yy	Addiere A zu yy	C und Z
4A yy	ORI yy	Bitweises Oder von A und yy	Z
4B yy	SBCI yy	$A := A - yy - C$	C und Z
4C yy	XORI yy	Bitweises ExOder von A und yy	Z
4D yy	ADCI yy	$A := A + yy + C$	C und Z
4E yy	ANDI yy	Bitweises Und von A und yy	Z
4F yy	SETFI yy	Setze C und Z entsprechend Bit 6/7 von yy und behalte A unverändert bei	C und Z

Dekodertabelle



Assembler-Aufgaben I

Assembler-Aufgaben I

1.: Bit x von I/O-Port y abfragen ($x = 6, y = 0x42$):

Assembler-Aufgaben I

1.: Bit x von I/O-Port y abfragen ($x = 6, y = 0x42$):

```
in    0x42
```

Assembler-Aufgaben I

1.: Bit x von I/O-Port y abfragen ($x = 6$, $y = 0x42$):

```
in      0x42  
andi   0x40
```

Assembler-Aufgaben I

1.: Bit x von I/O-Port y abfragen ($x = 6$, $y = 0x42$):

```
in      0x42
andi    0x40
beq     weitweg
```

Assembler-Aufgaben I

1.: Bit x von I/O-Port y abfragen ($x = 6, y = 0x42$):

:schleife

in 0x42

andi 0x40

beq schleife

. . .

:weitweg

. . .

Assembler-Aufgaben I

2.: Bit x_1 und x_2 von I/O-Port y setzen ($x_1 = 6$, $x_2 = 0$, $y = 0x2A$):

Assembler-Aufgaben I

2.: Bit x_1 und x_2 von I/O-Port y setzen ($x_1 = 6$, $x_2 = 0$, $y = 0x2A$):

```
out 0x2A
```

Assembler-Aufgaben I

2.: Bit x_1 und x_2 von I/O-Port y setzen ($x_1 = 6$, $x_2 = 0$, $y = 0x2A$):

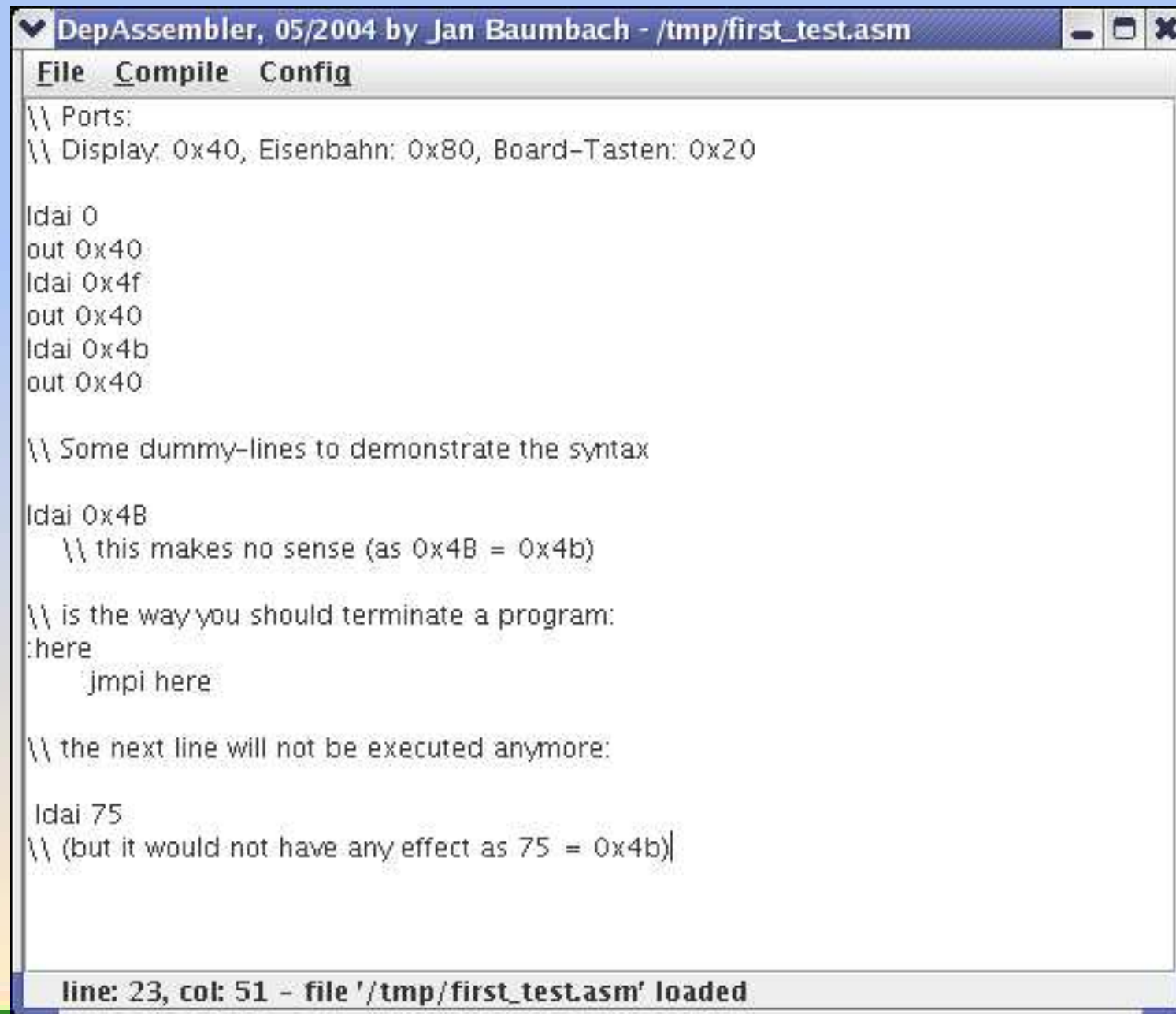
```
ldai 0x41
```

```
out 0x2A
```

```
. . .
```


DepAssemblerEditor

DepAssemblerEditor



The screenshot shows a window titled "DepAssembler, 05/2004 by Jan Baumbach - /tmp/first_test.asm". The window has a menu bar with "File", "Compile", and "Config". The main text area contains the following assembly code:

```
\\ Ports:
\\ Display: 0x40, Eisenbahn: 0x80, Board-Tasten: 0x20

ldai 0
out 0x40
ldai 0x4f
out 0x40
ldai 0x4b
out 0x40

\\ Some dummy-lines to demonstrate the syntax

ldai 0x4B
    \\ this makes no sense (as 0x4B = 0x4b)

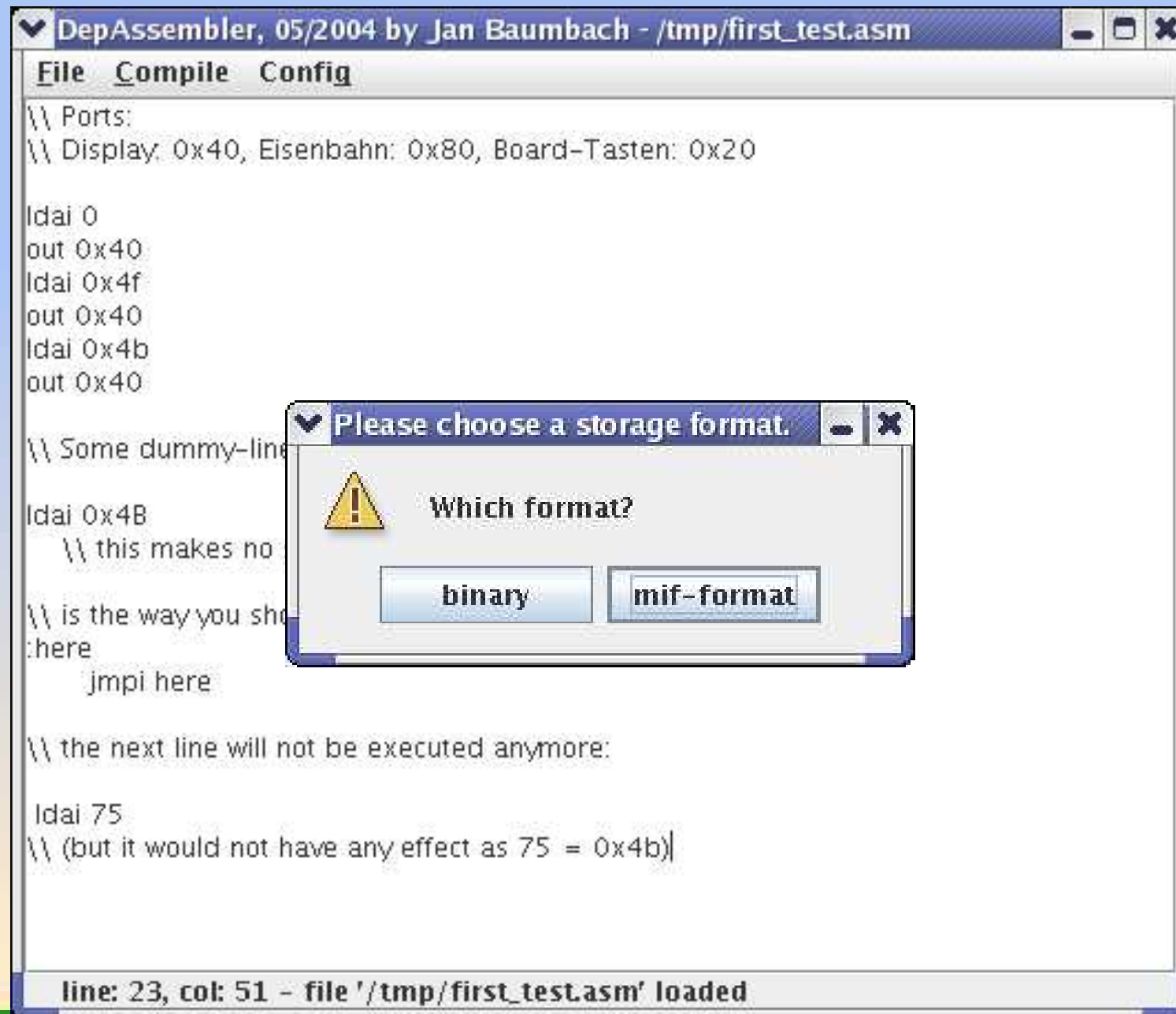
\\ is the way you should terminate a program:
:here
    jmp here

\\ the next line will not be executed anymore:

ldai 75
\\ (but it would not have any effect as 75 = 0x4b)
```

At the bottom of the window, a status bar displays the text: "line: 23, col: 51 - file '/tmp/first_test.asm' loaded".

DepAssemblerEditor



Assembler-Aufgaben II

Assembler-Aufgaben II

3.: Bit-Operationen / Bit-Masken:

Assembler-Aufgaben II

3.: Bit-Operationen / Bit-Masken:

```
andi    0x40 (einzelne Bits abfragen)
```

Assembler-Aufgaben II

3.: Bit-Operationen / Bit-Masken:

`andi 0x40` (einzelne Bits abfragen)

`andi 0xF5` (einzelne Bits löschen)

Assembler-Aufgaben II

3.: Bit-Operationen / Bit-Masken:

`andi 0x40` (einzelne Bits abfragen)

`andi 0xF5` (einzelne Bits löschen)

`ori 0x18` (einzelne Bits setzen)

Assembler-Aufgaben II

3.: Bit-Operationen / Bit-Masken:

`andi` `0x40` (einzelne Bits abfragen)
`andi` `0xF5` (einzelne Bits löschen)
`ori` `0x18` (einzelne Bits setzen)
`xori` `0x24` (einzelne Bits umkehren)

Assembler-Aufgaben II

4.: Vergleichsoperationen:

Assembler-Aufgaben II

4.: Vergleichsoperationen:

→ `andi` `0x40`

`beq` `ziel` (einzelne Bits abfragen)

Assembler-Aufgaben II

4.: Vergleichsoperationen:

andi 0x40

beq ziel (einzelne Bits abfragen)

→ setfi 0x00

sbc i 0x23

beq ziel (bestimmten Wert abfragen)

Assembler-Aufgaben II

4.: Vergleichsoperationen:

`andi 0x40`

`beq ziel` (einzelne Bits abfragen)

`setfi 0x00`

`sbc i 0x23`

`beq ziel` (bestimmten Wert abfragen)

→ `setfi 0x00`

`sbc i 23`

`bcs unter23` (Wertebereich abfragen)

Assembler-Aufgaben II

4.: Vergleichsoperationen:

`andi 0x40`

`beq ziel` (einzelne Bits abfragen)

`setfi 0x00`

`sbc i 0x23`

`beq ziel` (bestimmten Wert abfragen)

`setfi 0x00`

`sbc i 23`

`bcs unter23` (Wertebereich abfragen)

→ `setfi 0x00`

`adc i 23`

`bcs ueber232` (Wertebereich abfragen)

Assembler-Aufgaben II

4.: Vergleichsoperationen:

`andi 0x40`

`beq ziel` (einzelne Bits abfragen)

`setfi 0x00`

`sbc i 0x23`

`beq ziel` (bestimmten Wert abfragen)

`setfi 0x00`

`sbc i 23`

`bcs unter23` (Wertebereich abfragen)

→ `addi 23`

`bcs ueber232` (Wertebereich abfragen)

Assembler-Aufgaben II

4.: Vergleichsoperationen:

andi 0x40

beq ziel (einzelne Bits abfragen)

setfi 0x00

sbc i 0x23

beq ziel (bestimmten Wert abfragen)

setfi 0x00

sbc i 23

bcs unter23 (Wertebereich abfragen)

addi 23

bcs ueber232 (Wertebereich abfragen)

→ xori 0x24

beq ziel (bestimmten Wert abfragen ohne Carry)

Assembler-Aufgaben II

5.: Zwischenspeicher:

Assembler-Aufgaben II

5.: Zwischenspeicher:

```
. . .  
sta    0x00  
  
. . .  
lda    0x00  
  
. . .
```

Assembler-Aufgaben II

6.: Addition 16 Bit:

Assembler-Aufgaben II

6.: Addition 16 Bit:

Die 16 Bit-Zahl in den RAM-Zellen 0x03 und 0x04 (LSB in 0x03, MSB in 0x04) soll zu der Zahl in den Zellen 0x05 und 0x06 (LSB in 0x05, MSB in 0x06) addiert werden.

Das Ergebnis soll in den Zellen 0x07 und 0x08 gespeichert werden.

Assembler-Aufgaben II

6.: Addition 16 Bit:

Die 16 Bit-Zahl in den RAM-Zellen 0x03 und 0x04 (LSB in 0x03, MSB in 0x04) soll zu der Zahl in den Zellen 0x05 und 0x06 (LSB in 0x05, MSB in 0x06) addiert werden.

Das Ergebnis soll in den Zellen 0x07 und 0x08 gespeichert werden.

```
lda    0x03
add    0x05 (adc falls gewünscht)
sta    0x07
```

Assembler-Aufgaben II

6.: Addition 16 Bit:

Die 16 Bit-Zahl in den RAM-Zellen 0x03 und 0x04 (LSB in 0x03, MSB in 0x04) soll zu der Zahl in den Zellen 0x05 und 0x06 (LSB in 0x05, MSB in 0x06) addiert werden.

Das Ergebnis soll in den Zellen 0x07 und 0x08 gespeichert werden.

```
lda    0x03
add    0x05 (adc falls gewünscht)
sta    0x07

lda    0x04
adc    0x06 (adc notwendig!)
sta    0x08
```

Assembler-Aufgaben II

7.: Multiplikation:

Assembler-Aufgaben II

7.: Multiplikation:

Die 8 Bit-Zahl in der RAM-Zelle 0x03 soll mit dem Wert in Zelle 0x04 multipliziert werden. Das Ergebnis soll in den Zellen 0x05 und 0x06 (LSB in 0x05, MSB in 0x06) gespeichert werden.

Assembler-Aufgaben II

7.: Multiplikation:

Die 8 Bit-Zahl in der RAM-Zelle 0x03 soll mit dem Wert in Zelle 0x04 multipliziert werden. Das Ergebnis soll in den Zellen 0x05 und 0x06 (LSB in 0x05, MSB in 0x06) gespeichert werden.

```
ldai    0x00
```

```
sta     0x05
```

```
sta     0x06
```

Assembler-Aufgaben II

7.: Multiplikation:

Die 8 Bit-Zahl in der RAM-Zelle 0x03 soll mit dem Wert in Zelle 0x04 multipliziert werden. Das Ergebnis soll in den Zellen 0x05 und 0x06 (LSB in 0x05, MSB in 0x06) gespeichert werden.

```
ldai    0x00
sta      0x05
sta      0x06
lda      0x03
sta      0x00 (evtl. Original unverändert lassen)
```

Assembler-Aufgaben II

7.: Multiplikation:

Die 8 Bit-Zahl in der RAM-Zelle 0x03 soll mit dem Wert in Zelle 0x04 multipliziert werden. Das Ergebnis soll in den Zellen 0x05 und 0x06 (LSB in 0x05, MSB in 0x06) gespeichert werden.

```
:schleife    beq     Ende
              lda     0x04
              add     0x05 (kein adc)
              sta     0x05
              bcs     ueberlauf
:weiter      lda     0x00
              sbci    0x01
              sta     0x00
              jmp     schleife
:Ende        jmp     Ende
```

Assembler-Aufgaben II

7.: Multiplikation:

Die 8 Bit-Zahl in der RAM-Zelle 0x03 soll mit dem Wert in Zelle 0x04 multipliziert werden. Das Ergebnis soll in den Zellen 0x05 und 0x06 (LSB in 0x05, MSB in 0x06) gespeichert werden.

```
:ueberlauf  ldai    0x01
              add    0x06
              sta    0x06
              setfi  0x00
              jmp    weiter
```

Assembler-Aufgaben II

7.: Multiplikation:

Die 8 Bit-Zahl in der RAM-Zelle 0x03 soll mit dem Wert in Zelle 0x04 multipliziert werden. Das Ergebnis soll in den Zellen 0x05 und 0x06 (LSB in 0x05, MSB in 0x06) gespeichert werden.

```
:ueberlauf  ldai    0x01
              add    0x06
              sta    0x06
              jmp    weiter
```

Assembler-Aufgaben II

8.: Division:

Assembler-Aufgaben II

8.: Division:

??? (ähnlich wie Multiplikation!)

Assembler-Aufgaben II

9.: Links-Schieben:

Assembler-Aufgaben II

9.: Links-Schieben:

→ Multiplikation mit 2 → Addition

Assembler-Aufgaben II

10.: Links-Rotieren:

Assembler-Aufgaben II

10.: Links-Rotieren:

→ Carry auswerten (bcs), ggf. 1 addieren

Assembler-Aufgaben II

11.: Rechts-Rotieren:

Assembler-Aufgaben II

11.: Rechts-Rotieren:

1. Variante: Bits einzeln abfragen

Assembler-Aufgaben II

11.: Rechts-Rotieren:

1. Variante: Bits einzeln abfragen

2. Variante: 8-n Bits nach links rotieren

Assembler-Aufgaben II

12.: Rechts-Schieben:

Assembler-Aufgaben II

12.: Rechts-Schieben:

1. Variante: Bits einzeln abfragen

Assembler-Aufgaben II

12.: Rechts-Schieben:

1. Variante: Bits einzeln abfragen

2. Variante: Nach rechts rotieren und MSBs maskieren

Assembler-Aufgaben II

12.: Rechts-Schieben:

1. Variante: Bits einzeln abfragen

2. Variante: Nach rechts rotieren und MSBs maskieren

3. Variante: Rechts-Schieben = Division durch 2 \rightarrow DIV-Algorithmus

Assembler-Aufgaben III

Assembler-Aufgaben III

13.: Zur Übung einfache Aufgaben selbst lösen:

Assembler-Aufgaben III

13.: Zur Übung einfache Aufgaben selbst lösen:

- Inhalt der RAM-Zelle 0x03 mit Inhalt der RAM-Zelle 0x05 vergleichen. Ist der Wert an Adresse 0x03 größer oder gleich dem Wert an Adresse 0x05, dann eine 1 in RAM-Zelle 0x07 schreiben, andernfalls eine 2.

Assembler-Aufgaben III

13.: Zur Übung einfache Aufgaben selbst lösen:

- Inhalt der RAM-Zelle 0x03 mit Inhalt der RAM-Zelle 0x05 vergleichen. Ist der Wert an Adresse 0x03 größer oder gleich dem Wert an Adresse 0x05, dann eine 1 in RAM-Zelle 0x07 schreiben, andernfalls eine 2.
- Den selben Vergleich auf I/O-Ports durchführen

Assembler-Aufgaben III

13.: Zur Übung einfache Aufgaben selbst lösen:

- Inhalt der RAM-Zelle 0x03 mit Inhalt der RAM-Zelle 0x05 vergleichen. Ist der Wert an Adresse 0x03 größer oder gleich dem Wert an Adresse 0x05, dann eine 1 in RAM-Zelle 0x07 schreiben, andernfalls eine 2.
- Den selben Vergleich auf I/O-Ports durchführen
- Ein Operationsergebnis auf Adresse 0x07 ausgeben (SBC, ADD, MUL,...).

Assembler-Aufgaben III

13.: Zur Übung einfache Aufgaben selbst lösen:

- Inhalt der RAM-Zelle 0x03 mit Inhalt der RAM-Zelle 0x05 vergleichen. Ist der Wert an Adresse 0x03 größer oder gleich dem Wert an Adresse 0x05, dann eine 1 in RAM-Zelle 0x07 schreiben, andernfalls eine 2.
- Den selben Vergleich auf I/O-Ports durchführen
- Ein Operationsergebnis auf Adresse 0x07 ausgeben (SBC, ADD, MUL,...).
- Füllen Sie die RAM-Zellen 3 bis 90 mit den Zahlen 3 bis 90
("3" → 0x03 , "4" → 0x04 , ...)

Assembler-Aufgaben III

13.: Zur Übung einfache Aufgaben selbst lösen:

- Inhalt der RAM-Zelle 0x03 mit Inhalt der RAM-Zelle 0x05 vergleichen. Ist der Wert an Adresse 0x03 größer oder gleich dem Wert an Adresse 0x05, dann eine 1 in RAM-Zelle 0x07 schreiben, andernfalls eine 2.
- Den selben Vergleich auf I/O-Ports durchführen
- Ein Operationsergebnis auf Adresse 0x07 ausgeben (SBC, ADD, MUL,...).
- Füllen Sie die RAM-Zellen 3 bis 90 mit den Zahlen 3 bis 90
("3" → 0x03 , "4" → 0x04 , ...)
 - Die Zahl in einer RAM-Zelle um n Bit nach links schieben

Vorversuche

Vorversuche

Vorbereitungsaufgabe: Vorbereitung der DEP-CPU-Entwicklung

1. Machen Sie sich durch Studium des Theorieteils mit der Architektur und dem Befehlssatz der DEP-CPU vertraut.
2. Ergänzen Sie die Decodertabelle.
3. Entwerfen Sie zur Übung den Inkrementer "Inc + 1" (falls noch nicht in Versuch 5 geschehen) mit möglichst geringem Ressourcen-Verbrauch. Sie können die von Ihnen bereits entwickelten Blöcke "Halbaddierer" und "Volladdierer" wiederverwenden. Erstellen Sie den Entwurf zunächst auf Papier.

Versuche

Versuche

Praktikumsaufgabe: Entwicklung und Test der DEP-CPU

1. Machen Sie sich durch Studium der Unterlagen (im Praktikumsraum vorhanden) mit dem FPGA-Board und der Software "Quartus II" vertraut.
2. Entwerfen und testen Sie den gesamten Prozessor. Beginnen Sie zur Übung zweckmäßig mit dem Inkrementer "Inc + 1", erstellen Sie danach die weiteren kleinen Baugruppen (Multiplexer, Register) - sofern dies noch nicht in Versuch 5 geschehen ist. Dann können Sie sich die ALU vornehmen und schließlich den Decoder. Sie erhalten als Basis wieder eine Projektdatei von Ihrem Tutor.
Erstellen Sie Ihre CPU dabei nur innerhalb der "CPU"-Schematic-Datei bzw. in darin eingefügten Blöcken (= Dateien).
3. Zum Test können Sie wieder den Simulator in der Entwicklungsumgebung "Quartus II" verwenden. Sollte Ihr Design im Simulator fehlerfrei sein, nicht jedoch auf dem FPGA-Board laufen, so achten Sie auf die Einstellung "Timing"-Simulation im "Simulator-Tool" (siehe Quartus-Kurzbeschreibung). Diese Simulation ist zuverlässiger als die einfache "Functional"-Simulation.
4. Um die Funktionsfähigkeit Ihres Prozessors nachzuweisen, wird Ihnen von Ihrem Tutor ein Beispielprogramm vorgegeben, welches fehlerfrei abgearbeitet werden muss.

5. Schreiben Sie ein Assembler-Programm, das – ähnlich der Modelleisenbahn-Automaten-Aufgabe aus dem letzten Versuch – einen Zug mit *drei* Waggons zum Entkupppler zieht und dann nacheinander die Waggons abkuppelt und auf die Gleise 1 bis 3 verteilt.

Sie können über einen IN-Befehl an Adresse 80 (hexadezimal) die Reed-Kontakte abfragen und über einen OUT-Befehl (gleiche Adresse) Fahrstrom, Fahrtrichtung, den Entkupppler und die Weiche steuern. Lesen Sie hierzu nochmals die Beschreibung zum letzten Versuch.

Die Bits sind folgendermaßen angeordnet:

D7	D6	D5	D4	D3	D2	D1	D0
res.	res.	R345	R5	R4	R3	R2	R1

Lesen

D7	D6	D5	D4	D3	D2	D1	D0
res.	res.	res.	W2	W1	ENTK	RICH	FS

Schreiben

(res. = reserviert: Sowohl “High” als auch “Low” möglich)

6. Fragen Sie zum Erstellen und Übersetzen des Assembler-Programms Ihren Tutor. Er wird das Programm auch prüfen und mit Ihnen zusammen die Kompilierung Ihrer CPU und Konfiguration des FPGA-Boards vornehmen.

7. Bauen Sie die Eisenbahn in Absprache mit dem Tutor auf und testen Sie Ihr Programm.
8. Welche Modifikationen sind beim Entwurf einer 16-Bit CPU im Blockschaltbild der 8-Bit CPU bei Beibehaltung des 8-Bit Datenbusses und des Instruktionssatzes notwendigerweise vorzunehmen?
Geben Sie sinnvolle Ergänzungen des Instruktionssatzes und die dafür benötigten Erweiterungen der zuvor modifizierten CPU-Stuktur an.

[illegible]