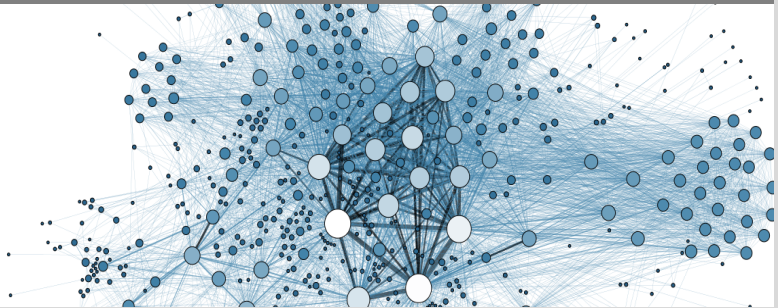
Karlsruher Institut für Technologie

lukasbach

Grundbegriffe der Informatik
Tutorium 33

Lukas Bach, lukas.bach@student.kit.edu | 1.12.2016



Lukas Bach, lukas.bach@student.kit.edu

Anmerkungen zum letzten Übungsblatt

Lukas Bach, lukas.bach@student.kit.edu

Anmerkungen zum letzten Übungsblatt

- Was ist sind die folgenden Mengen?

Lukas Bach, lukas.bach@student.kit.edu

Anmerkungen zum letzten Übungsblatt

- Was ist sind die folgenden Mengen?
  - $\mathbb{N}$

Lukas Bach, lukas.bach@student.kit.edu

Anmerkungen zum letzten Übungsblatt

- Was ist sind die folgenden Mengen?
  - $\mathbb{N}$ = Menge der natürlichen Zahlen (1, 2, 3, ...)

Lukas Bach, lukas.bach@student.kit.edu

Anmerkungen zum letzten Übungsblatt

- Was ist sind die folgenden Mengen?
  - $\mathbb{N}$ = Menge der natürlichen Zahlen (1, 2, 3, ...)
  - $\mathbb{N}_0$

Lukas Bach, lukas.bach@student.kit.edu

Anmerkungen zum letzten Übungsblatt

- Was ist sind die folgenden Mengen?
  - $\mathbb{N}$ = Menge der natürlichen Zahlen (1, 2, 3, ...)
  - $\mathbb{N}_0 = \mathbb{N} \cup \{0\}$

Lukas Bach, lukas.bach@student.kit.edu

Anmerkungen zum letzten Übungsblatt

- Was ist sind die folgenden Mengen?
  - $\mathbb{N}$ = Menge der natürlichen Zahlen (1, 2, 3, ...)
  - $\mathbb{N}_0 = \mathbb{N} \cup \{0\}$
  - $\mathbb{R}$

Lukas Bach, lukas.bach@student.kit.edu

Anmerkungen zum letzten Übungsblatt

- Was ist sind die folgenden Mengen?
  - $\mathbb{N}$ = Menge der natürlichen Zahlen (1, 2, 3, ...)
  - $\mathbb{N}_0 = \mathbb{N} \cup \{0\}$
  - $\mathbb{R}$ = Menge der Reellen Zahlen

Lukas Bach, lukas.bach@student.kit.edu

Anmerkungen zum letzten Übungsblatt

- Was ist sind die folgenden Mengen?
  - $\mathbb{N}$ = Menge der natürlichen Zahlen (1, 2, 3, ...)
  - $\mathbb{N}_0 = \mathbb{N} \cup \{0\}$
  - $\mathbb{R}$ = Menge der Reellen Zahlen
  - $\mathbb{R}^+$

Lukas Bach, lukas.bach@student.kit.edu

Anmerkungen zum letzten Übungsblatt

- Was ist sind die folgenden Mengen?
  - $\mathbb{N}$ = Menge der natürlichen Zahlen (1, 2, 3, ...)
  - $\mathbb{N}_0 = \mathbb{N} \cup \{0\}$
  - $\mathbb{R}$ = Menge der Reellen Zahlen
  - $\mathbb{R}^+$ = Menge der positiven reellen Zahlen

Lukas Bach, lukas.bach@student.kit.edu

Anmerkungen zum letzten Übungsblatt

- Was ist sind die folgenden Mengen?
  - $\mathbb{N}$ = Menge der natürlichen Zahlen (1, 2, 3, ...)
  - $\mathbb{N}_0 = \mathbb{N} \cup \{0\}$
  - $\mathbb{R}$ = Menge der Reellen Zahlen
  - $\mathbb{R}^+$ = Menge der positiven reellen Zahlen
  - $\mathbb{R}_0$

Lukas Bach, lukas.bach@student.kit.edu

Anmerkungen zum letzten Übungsblatt

- Was ist sind die folgenden Mengen?
  - $\mathbb{N}$ = Menge der natürlichen Zahlen (1, 2, 3, ...)
  - $\mathbb{N}_0 = \mathbb{N} \cup \{0\}$
  - $\mathbb{R}$ = Menge der Reellen Zahlen
  - $\mathbb{R}^+$ = Menge der positiven reellen Zahlen
  - $\mathbb{R}_0$ gibt es nicht! 0 ist auch so schon in $\mathbb{R}$

Lukas Bach, lukas.bach@student.kit.edu

Anmerkungen zum letzten Übungsblatt

- Was ist sind die folgenden Mengen?
  - $\mathbb{N}$ = Menge der natürlichen Zahlen (1, 2, 3, ...)
  - $\mathbb{N}_0 = \mathbb{N} \cup \{0\}$
  - $\mathbb{R}$ = Menge der Reellen Zahlen
  - $\mathbb{R}^+$ = Menge der positiven reellen Zahlen
  - $\mathbb{R}_0$ gibt es nicht! 0 ist auch so schon in $\mathbb{R}$
  - $\mathbb{R}_0^+$ genauso nicht!

Lukas Bach, lukas.bach@student.kit.edu

Anmerkungen zum letzten Übungsblatt

- Was ist sind die folgenden Mengen?
  - $\mathbb{N}$ = Menge der natürlichen Zahlen (1, 2, 3, ...)
  - $\mathbb{N}_0 = \mathbb{N} \cup \{0\}$
  - $\mathbb{R}$ = Menge der Reellen Zahlen
  - $\mathbb{R}^+$ = Menge der positiven reellen Zahlen
  - $\mathbb{R}_0$ gibt es nicht! 0 ist auch so schon in $\mathbb{R}$
  - $\mathbb{R}_0^+$ genauso nicht!
- Aufgabe: $R : A^* \to A^*$
  - $R(\varepsilon) = \varepsilon$
  - $\forall x \in A : R(x) = x$
  - $\forall w \in A^* \forall x \in A \forall y \in A : R(xwy) = yR(w)x$
  - Zeige: $\forall n \in \mathbb{N}_0 : \forall w \in A^n : |R(w)| = |w|$

Lukas Bach, lukas.bach@student.kit.edu

Was ist die MIMA?

Lukas Bach, lukas.bach@student.kit.edu

Was ist die MIMA?

- Theoretischer, idealisierter Prozessor

Lukas Bach, lukas.bach@student.kit.edu

Was ist die MIMA?

- Theoretischer, idealisierter Prozessor
- Funktioniert wie ein echter Prozessor, ist aber simpler

Lukas Bach, lukas.bach@student.kit.edu

Was ist die MIMA?

- Theoretischer, idealisierter Prozessor
- Funktioniert wie ein echter Prozessor, ist aber simpler
- Nah an Technischer Informatik

Lukas Bach, lukas.bach@student.kit.edu

Was ist die MIMA?

- Theoretischer, idealisierter Prozessor
- Funktioniert wie ein echter Prozessor, ist aber simpler
- Nah an Technischer Informatik

Grundaufbau:

Lukas Bach, lukas.bach@student.kit.edu

Was ist die MIMA?

- Theoretischer, idealisierter Prozessor
- Funktioniert wie ein echter Prozessor, ist aber simpler
- Nah an Technischer Informatik

Grundaufbau:

- Adressen als 20*bit* Datenwort

Lukas Bach, lukas.bach@student.kit.edu

Was ist die MIMA?

- Theoretischer, idealisierter Prozessor
- Funktioniert wie ein echter Prozessor, ist aber simpler
- Nah an Technischer Informatik

Grundaufbau:

- Adressen als 20*bit* Datenwort
- Speicherworte als 24*bit* Datenwort

Lukas Bach, lukas.bach@student.kit.edu

Was ist die MIMA?

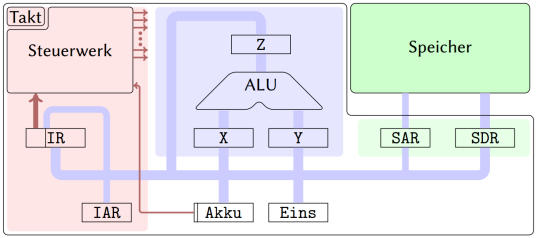
- Theoretischer, idealisierter Prozessor
- Funktioniert wie ein echter Prozessor, ist aber simpler
- Nah an Technischer Informatik

Grundaufbau:

- Adressen als 20*bit* Datenwort
- Speicherworte als 24*bit* Datenwort
- Maschinenbefehle als...
  - 4*bit* Befehl und 20*bit* Adresse
  - oder 8*bit* Befehl und unwichtigem Rest

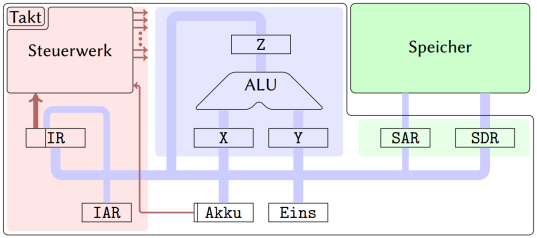
Lukas Bach, lukas.bach@student.kit.edu

Aufbau der MIMA: Steuerwerk



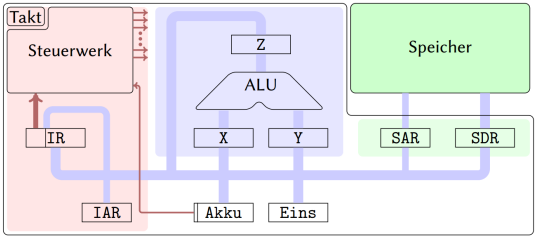
Lukas Bach, lukas.bach@student.kit.edu

Aufbau der MIMA: Steuerwerk



Steuerwerk

Aufbau der MIMA: Steuerwerk

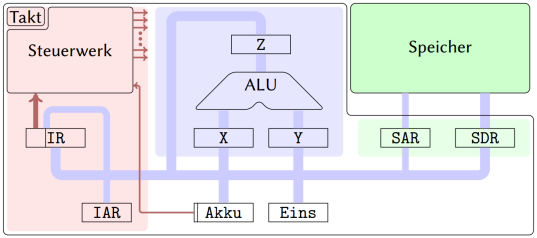


Steuerwerk

- Instruction Register (IR) enthält den nächsten auszuführenden Befehl

Lukas Bach, lukas.bach@student.kit.edu

Aufbau der MIMA: Steuerwerk

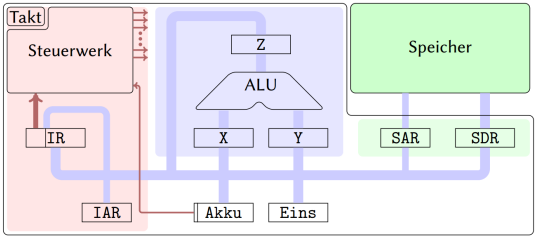


Steuerwerk

- Instruction Register (IR) enthält den nächsten auszuführenden Befehl
- Instruction Adress Register (IAR) enthält die Adresse des nächsten Befehls

Lukas Bach, lukas.bach@student.kit.edu

Aufbau der MIMA: Steuerwerk

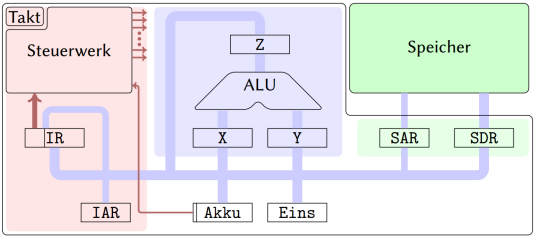


Steuerwerk

- Instruction Register (IR) enthält den nächsten auszuführenden Befehl
- Instruction Adress Register (IAR) enthält die Adresse des nächsten Befehls
- Takt bestimmt die “Tickrate”, also die Geschwindigkeit

Lukas Bach, lukas.bach@student.kit.edu

Aufbau der MIMA: Steuerwerk

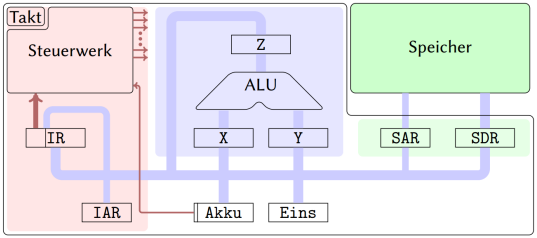


Steuerwerk

- Instruction Register (IR) enthält den nächsten auszuführenden Befehl
- Instruction Adress Register (IAR) enthält die Adresse des nächsten Befehls
- Takt bestimmt die “Tickrate”, also die Geschwindigkeit
- Steuerwerk interpretiert alle Befehle und führt sie aus

Lukas Bach, lukas.bach@student.kit.edu

Aufbau der MIMA: Steuerwerk

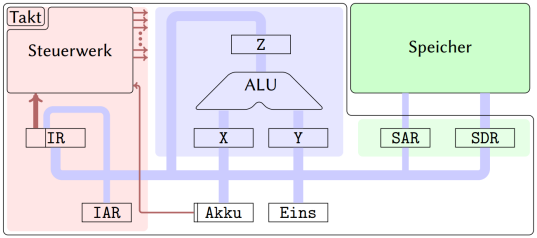


Steuerwerk

- Instruction Register (IR) enthält den nächsten auszuführenden Befehl
- Instruction Adress Register (IAR) enthält die Adresse des nächsten Befehls
- Takt bestimmt die “Tickrate”, also die Geschwindigkeit
- Steuerwerk interpretiert alle Befehle und führt sie aus
- Welche Befehle es gibt: Siehe später

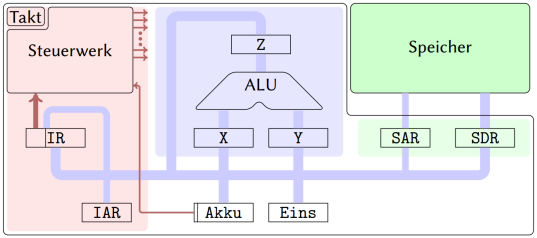
Lukas Bach, lukas.bach@student.kit.edu

Aufbau der MIMA: Akku und Eins



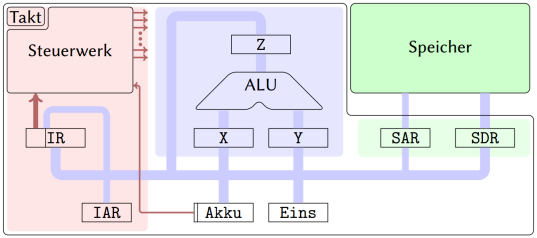
Lukas Bach, lukas.bach@student.kit.edu

Aufbau der MIMA: Akku und Eins



Akku und Eins

Aufbau der MIMA: Akku und Eins

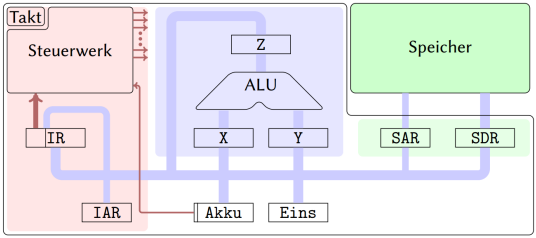


Akku und Eins

- Akku dient als Zwischenspeicher für Datenworte

Lukas Bach, lukas.bach@student.kit.edu

Aufbau der MIMA: Akku und Eins

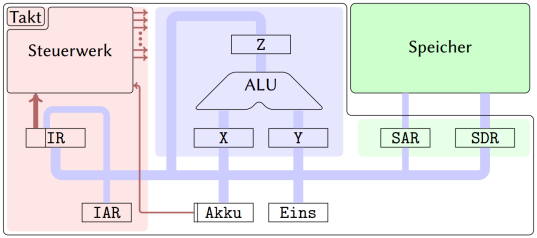


Akku und Eins

- Akku dient als Zwischenspeicher für Datenworte
- Hält maximal ein Wort

Lukas Bach, lukas.bach@student.kit.edu

Aufbau der MIMA: Akku und Eins

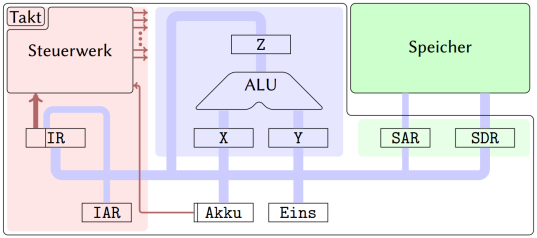


Akku und Eins

- Akku dient als Zwischenspeicher für Datenworte
- Hält maximal ein Wort
- Eins liefert die Konstante 1, hält also Strom

Lukas Bach, lukas.bach@student.kit.edu

Aufbau der MIMA: Akku und Eins

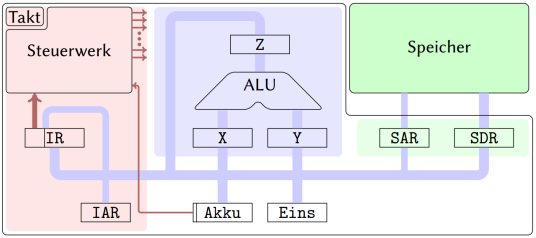


Akku und Eins

- Akku dient als Zwischenspeicher für Datenworte
- Hält maximal ein Wort
- Eins liefert die Konstante 1, hält also Strom
- z.B. erhöhen des IAR

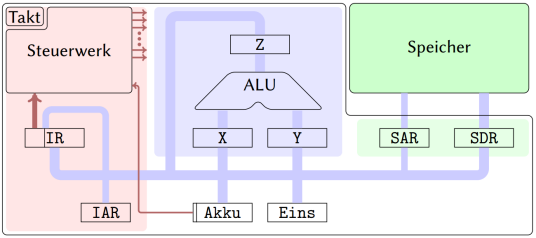
Lukas Bach, lukas.bach@student.kit.edu

Aufbau der MIMA: ALU



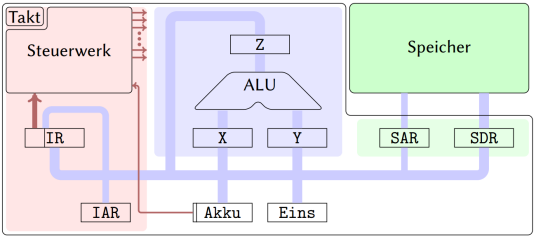
Lukas Bach, lukas.bach@student.kit.edu

Aufbau der MIMA: ALU



Arithmetic Logic Unit (ALU) / Rechenwerk

Aufbau der MIMA: ALU

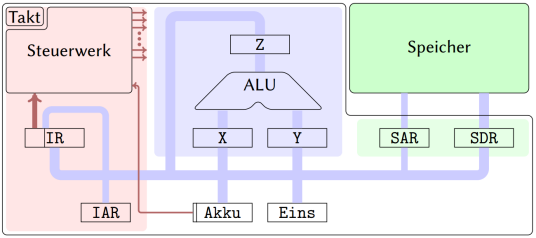


Arithmetic Logic Unit (ALU) / Rechenwerk

- Durchführt arithmetische Operationen

Lukas Bach, lukas.bach@student.kit.edu

Aufbau der MIMA: ALU

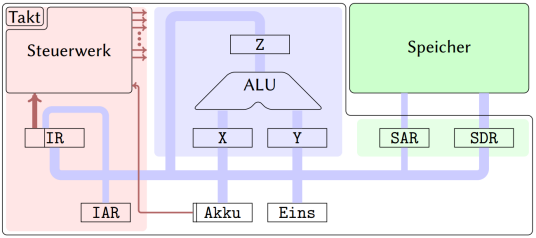


Arithmetic Logic Unit (ALU) / Rechenwerk

- Durchführt arithmetische Operationen
- **mod** , **div** , $+$, $-$, ..., bitweises OR/AND/...

Lukas Bach, lukas.bach@student.kit.edu

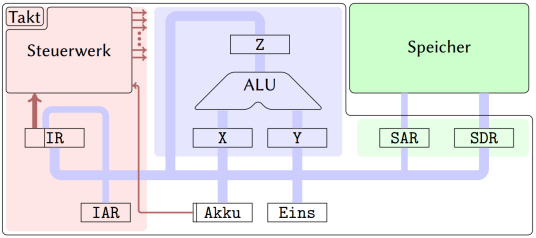
Aufbau der MIMA: ALU



Arithmetic Logic Unit (ALU) / Rechenwerk

- Durchführt arithmetische Operationen
- **mod** , **div** , $+$, $-$, ..., bitweises OR/AND/...
- X und Y sind Eingaberegister

Aufbau der MIMA: ALU

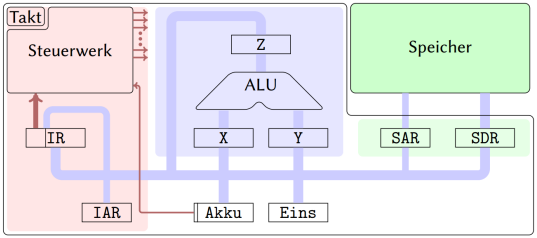


Arithmetic Logic Unit (ALU) / Rechenwerk

- Durchführt arithmetische Operationen
- **mod** , **div** , $+$, $-$, ..., bitweises OR/AND/...
- X und Y sind Eingaberegister
- Z ist Ausgaberegister

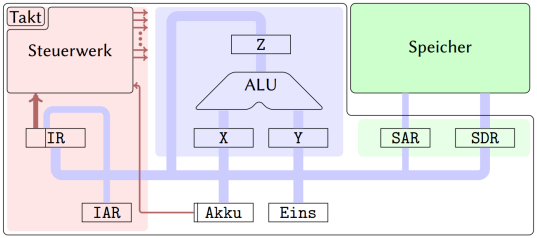
Lukas Bach, lukas.bach@student.kit.edu

Aufbau der MIMA: ALU



Lukas Bach, lukas.bach@student.kit.edu

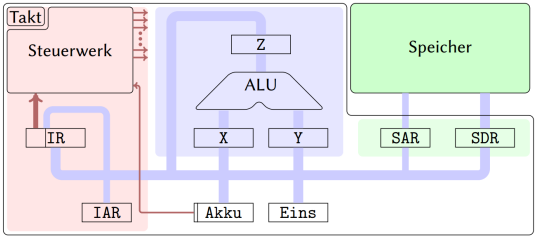
Aufbau der MIMA: ALU



Speicher(werk)

Speicher selbst speichert Befehle und Daten.

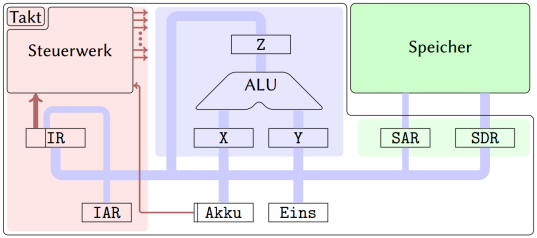
Aufbau der MIMA: ALU



Speicher(werk)

Speicher selbst speichert Befehle und Daten. Speicherwerk besteht aus:

Aufbau der MIMA: ALU

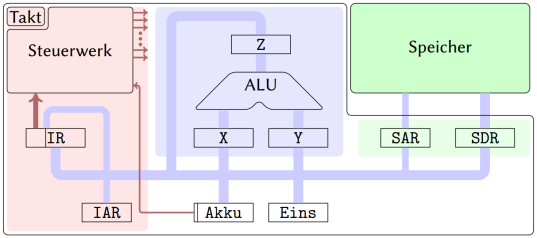


Speicher(werk)

Speicher selbst speichert Befehle und Daten. Speicherwerk besteht aus:

- Speicheradressregister (SAR)
  ist die Adresse, bei der im Speicher gespeichert/gelesen werden soll

Aufbau der MIMA: ALU



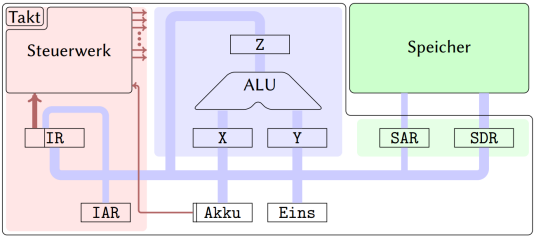
Speicher(werk)

Speicher selbst speichert Befehle und Daten. Speicherwerk besteht aus:

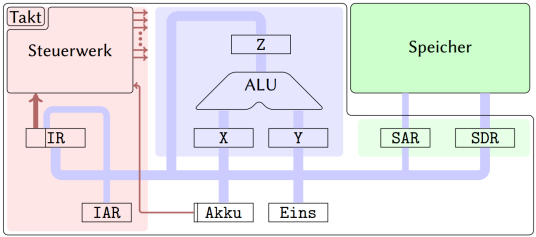
- Speicheradressregister (SAR) ist die Adresse, bei der im Speicher gespeichert/gelesen werden soll
- Speicherdatenregister (SDR) Datum, das bei der Adresse gespeichert werden soll/ gelesen wurde.

Lukas Bach, lukas.bach@student.kit.edu

Aufbau der MIMA: ALU

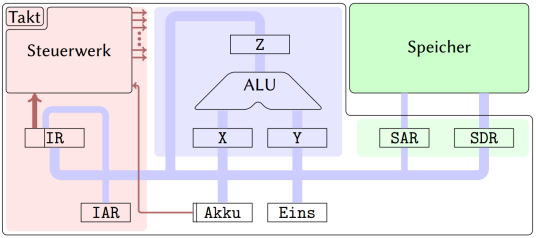


Aufbau der MIMA: ALU



Busse

Aufbau der MIMA: ALU

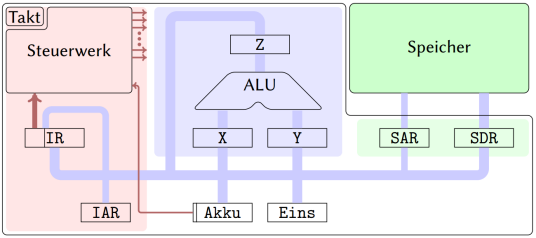


Busse

- “Kabel” zwischen den Verbindungen

Lukas Bach, lukas.bach@student.kit.edu

Aufbau der MIMA: ALU

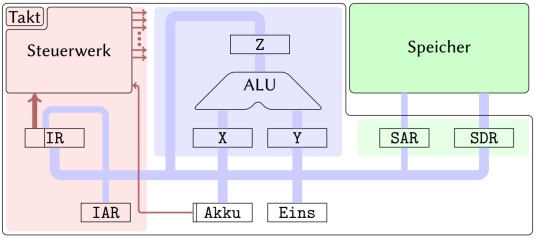


Busse

- “Kabel” zwischen den Verbindungen
- Ein kompletter Bus überträgt entweder 1, 0, oder nichts

Lukas Bach, lukas.bach@student.kit.edu

Aufbau der MIMA: ALU



Busse

- “Kabel” zwischen den Verbindungen
- Ein kompletter Bus überträgt entweder 1, 0, oder nichts
- Kann nur eine einzige Information auf einmal übertragen

Lukas Bach, lukas.bach@student.kit.edu

Konventionen zu MIMA Programmen

Um MIMA Programme und dazugehörige Definitionen verständlicher zu machen, vereinbaren wir folgende Konventionen:

Lukas Bach, lukas.bach@student.kit.edu

Konventionen zu MIMA Programmen

Um MIMA Programme und dazugehörige Definitionen verständlicher zu machen, vereinbaren wir folgende Konventionen:

- Befehle (eigentlich Bitfolge) schreiben wir als Befehlname und Adresse

Lukas Bach, lukas.bach@student.kit.edu

Konventionen zu MIMA Programmen

Um MIMA Programme und dazugehörige Definitionen verständlicher zu machen, vereinbaren wir folgende Konventionen:

- Befehle (eigentlich Bitfolge) schreiben wir als Befehlname und Adresse
  - $0010000000000000000101010 \equiv \mathit{STV}\ 42$

Lukas Bach, lukas.bach@student.kit.edu

Konventionen zu MIMA Programmen

Um MIMA Programme und dazugehörige Definitionen verständlicher zu machen, vereinbaren wir folgende Konventionen:

- Befehle (eigentlich Bitfolge) schreiben wir als Befehlname und Adresse
  - 001000000000000000101010 $\equiv$ *STV* 42
- $X \leftarrow Y \equiv$ “Der Variable X wird der Wert Y zugewiesen”

Lukas Bach, lukas.bach@student.kit.edu

MIMA Befehle

Eine MIMA-Maschine beherrscht folgende Maschinenbefehle:

Befehlssyntax	Formel	Bedeutung

Lukas Bach, lukas.bach@student.kit.edu

MIMA Befehle

Eine MIMA-Maschine beherrscht folgende Maschinenbefehle:

Befehlssyntax	Formel	Bedeutung
LDC const	*Akku* $\leftarrow$ *const*	Lade eine Konstate *const* in den Akku

MIMA Befehle

Eine MIMA-Maschine beherrscht folgende Maschinenbefehle:

Befehlssyntax	Formel	Bedeutung
LDC const	$Akku \leftarrow const$	Lade eine Konstate *const* in den Akku
LDV adr	$Akku \leftarrow M(adr)$	Lade einen Wert vom Speicher bei Adresse *adr* in den Akku

MIMA Befehle

Eine MIMA-Maschine beherrscht folgende Maschinenbefehle:

Befehlssyntax	Formel	Bedeutung
LDC const	$Akku \leftarrow const$	Lade eine Konstate *const* in den Akku
LDV adr	$Akku \leftarrow M(adr)$	Lade einen Wert vom Speicher bei Adresse *adr* in den Akku
STV adr	$M(adr) \leftarrow Akku$	Lade Speichere den Wert aus dem Akku im Speicher bei Adresse *adr*

MIMA Befehle

Eine MIMA-Maschine beherrscht folgende Maschinenbefehle:

Befehlssyntax	Formel	Bedeutung
LDC const	$Akku \leftarrow const$	Lade eine Konstate *const* in den Akku
LDV adr	$Akku \leftarrow M(adr)$	Lade einen Wert vom Speicher bei Adresse *adr* in den Akku
STV adr	$M(adr) \leftarrow Akku$	Lade Speichere den Wert aus dem Akku im Speicher bei Adresse *adr*
LDIV adr	$Akku \leftarrow M(M(adr))$	Lade einen Wert vom Speicher bei der Adresse, die bei *adr* gespeichert ist, und lade den Wert in den Akku

MIMA Befehle

Eine MIMA-Maschine beherrscht folgende Maschinenbefehle:

Befehlssyntax	Formel	Bedeutung
LDC const	$Akku \leftarrow const$	Lade eine Konstate *const* in den Akku
LDV adr	$Akku \leftarrow M(adr)$	Lade einen Wert vom Speicher bei Adresse *adr* in den Akku
STV adr	$M(adr) \leftarrow Akku$	Lade Speichere den Wert aus dem Akku im Speicher bei Adresse *adr*
LDIV adr	$Akku \leftarrow M(M(adr))$	Lade einen Wert vom Speicher bei der Adresse, die bei *adr* gespeichert ist, und lade den Wert in den Akku
STIV adr	$M(M(adr)) \leftarrow Akku$	Speichere den Wert im Akku bei der Adresse, die in *adr* gespeichert ist.

Lukas Bach, lukas.bach@student.kit.edu

MIMA Befehle (2)

Eine MIMA-Maschine beherrscht folgende Maschinenbefehle:

Befehlssyntax	Formel	Bedeutung

Lukas Bach, lukas.bach@student.kit.edu

MIMA Befehle (2)

Eine MIMA-Maschine beherrscht folgende Maschinenbefehle:

Befehlssyntax	Formel	Bedeutung
ADD adr	$Akku \leftarrow Akku + M(adr)$	Addiere den Wert bei *adr* zum Akku dazu.

Lukas Bach, lukas.bach@student.kit.edu

MIMA Befehle (2)

Eine MIMA-Maschine beherrscht folgende Maschinenbefehle:

Befehlssyntax	Formel	Bedeutung
ADD adr	$Akku \leftarrow Akku + M(adr)$	Addiere den Wert bei *adr* zum Akku dazu.
"*OP*" *adr*	$Akku$ "OP" $M(adr)$	Wende bitweise Operation auf Akku mit Wert bei *adr* an. $Op \in \{AND, OR, XOR\}$.

Lukas Bach, lukas.bach@student.kit.edu

MIMA Befehle (3)

Eine MIMA-Maschine beherrscht folgende Maschinenbefehle:

Befehlssyntax	Bedeutung

Lukas Bach, lukas.bach@student.kit.edu

MIMA Befehle (3)

Eine MIMA-Maschine beherrscht folgende Maschinenbefehle:

Befehlssyntax	Bedeutung
NOT	Bitweise Invertierung aller Bits des Akku-Datenwortes

Lukas Bach, lukas.bach@student.kit.edu

MIMA Befehle (3)

Eine MIMA-Maschine beherrscht folgende Maschinenbefehle:

Befehlssyntax	Bedeutung
NOT	Bitweise Invertierung aller Bits des Akku-Datenwortes
RAR	Rotiere alle Akku-Bits eins nach rechts

Lukas Bach, lukas.bach@student.kit.edu

MIMA Befehle (3)

Eine MIMA-Maschine beherrscht folgende Maschinenbefehle:

Befehlssyntax	Bedeutung
NOT	Bitweise Invertierung aller Bits des Akku-Datenwortes
RAR	Rotiere alle Akku-Bits eins nach rechts
EQL adr	Setze Akku auf $11\cdots11$, falls Wert bei *adr* gleich Akku-Wert, setze Akku auf $00\cdots00$ sonst.

Lukas Bach, lukas.bach@student.kit.edu

MIMA Befehle (3)

Eine MIMA-Maschine beherrscht folgende Maschinenbefehle:

Befehlssyntax	Bedeutung
NOT	Bitweise Invertierung aller Bits des Akku-Datenwortes
RAR	Rotiere alle Akku-Bits eins nach rechts
EQL adr	Setze Akku auf $11\cdots11$, falls Wert bei *adr* gleich Akku-Wert, setze Akku auf $00\cdots00$ sonst.
JMP adr	Springe zu Befehlsadresse *adr*

Lukas Bach, lukas.bach@student.kit.edu

MIMA Befehle (3)

Eine MIMA-Maschine beherrscht folgende Maschinenbefehle:

Befehlssyntax	Bedeutung
NOT	Bitweise Invertierung aller Bits des Akku-Datenwortes
RAR	Rotiere alle Akku-Bits eins nach rechts
EQL adr	Setze Akku auf $11\cdots11$, falls Wert bei *adr* gleich Akku-Wert, setze Akku auf $00\cdots00$ sonst.
JMP adr	Springe zu Befehlsadresse *adr*
JMN adr	Springe zu Befehlsadresse *adr*, falls Akku negativ (also erstes Bit $= 1$), sonst fahre normal fort.

Lukas Bach, lukas.bach@student.kit.edu

MIMA Befehle: Sichern und Laden

Lukas Bach, lukas.bach@student.kit.edu

MIMA Befehle: Sichern und Laden

- Befehle zum laden und Speichern in den Speicher

Lukas Bach, lukas.bach@student.kit.edu

MIMA Befehle: Sichern und Laden

- Befehle zum laden und Speichern in den Speicher
- LDV um Daten vom Speicher zu laden, STV um Daten in den Speicher zu schreiben

Lukas Bach, lukas.bach@student.kit.edu

MIMA Befehle: Sichern und Laden

- Befehle zum laden und Speichern in den Speicher
- LDV um Daten vom Speicher zu laden, STV um Daten in den Speicher zu schreiben
- LDC um eine Konstante zu laden

Lukas Bach, lukas.bach@student.kit.edu

MIMA Befehle: Sichern und Laden

- Befehle zum laden und Speichern in den Speicher
- LDV um Daten vom Speicher zu laden, STV um Daten in den Speicher zu schreiben
- LDC um eine Konstante zu laden
- Daten werden in einem Zwischenspeicher gelagert, der nur ein Datenwort hält

Lukas Bach, lukas.bach@student.kit.edu

MIMA Befehle: Sichern und Laden

- Befehle zum laden und Speichern in den Speicher
- LDV um Daten vom Speicher zu laden, STV um Daten in den Speicher zu schreiben
- LDC um eine Konstante zu laden
- Daten werden in einem Zwischenspeicher gelagert, der nur ein Datenwort hält: Akku.

Lukas Bach, lukas.bach@student.kit.edu

MIMA Befehle: Sichern und Laden

- Befehle zum laden und Speichern in den Speicher
- LDV um Daten vom Speicher zu laden, STV um Daten in den Speicher zu schreiben
- LDC um eine Konstante zu laden
- Daten werden in einem Zwischenspeicher gelagert, der nur ein Datenwort hält: Akku.

Beispiele:

MIMA Befehle: Sichern und Laden

- Befehle zum laden und Speichern in den Speicher
- LDV um Daten vom Speicher zu laden, STV um Daten in den Speicher zu schreiben
- LDC um eine Konstante zu laden
- Daten werden in einem Zwischenspeicher gelagert, der nur ein Datenwort hält: Akku.

Beispiele:

- *LDV* 9 lädt das Datum, das im Speicher bei Adresse 9 liegt, in den Akku.

Lukas Bach, lukas.bach@student.kit.edu

MIMA Befehle: Sichern und Laden

- Befehle zum laden und Speichern in den Speicher
- LDV um Daten vom Speicher zu laden, STV um Daten in den Speicher zu schreiben
- LDC um eine Konstante zu laden
- Daten werden in einem Zwischenspeicher gelagert, der nur ein Datenwort hält: Akku.

Beispiele:

- *LDV* 9 lädt das Datum, das im Speicher bei Adresse 9 liegt, in den Akku.
- *STV* 9 speichert das Datum, das im Akku liegt, in den Speicher an Adresse 9.

MIMA Befehle: Sichern und Laden

- Befehle zum laden und Speichern in den Speicher
- LDV um Daten vom Speicher zu laden, STV um Daten in den Speicher zu schreiben
- LDC um eine Konstante zu laden
- Daten werden in einem Zwischenspeicher gelagert, der nur ein Datenwort hält: Akku.

Beispiele:

- *LDV* 9 lädt das Datum, das im Speicher bei Adresse 9 liegt, in den Akku.
- *STV* 9 speichert das Datum, das im Akku liegt, in den Speicher an Adresse 9.
- *LDC* 4 lädt die Zahl 4 in den Akku (also kein Speicherzugriff).

MIMA Befehle: Sichern und Laden

Befehlssyntax	Formel	Bedeutung
LDC const	$Akku \leftarrow const$	Lade eine Konstate *const* in den Akku
LDV adr	$Akku \leftarrow M(adr)$	Lade einen Wert vom Speicher bei Adresse *adr* in den Akku
STV adr	$M(adr) \leftarrow Akku$	Lade Speichere den Wert aus dem Akku im Speicher bei Adresse *adr*

Lukas Bach, lukas.bach@student.kit.edu

MIMA Befehle: Sichern und Laden

Befehlssyntax	Formel	Bedeutung
LDC const	$Akku \leftarrow const$	Lade eine Konstate *const* in den Akku
LDV adr	$Akku \leftarrow M(adr)$	Lade einen Wert vom Speicher bei Adresse *adr* in den Akku
STV adr	$M(adr) \leftarrow Akku$	Lade Speichere den Wert aus dem Akku im Speicher bei Adresse *adr*

Beispielprogramm mit initialem Speicherabbild

LDC 5
STV a_1
LDC 7
STV a_2
⋮

⋮
LDV a_1
STV a_3
HALT

Adresse	Wert
a_1	0
a_2	0
a_3	0

MIMA Befehle: Indirektes Sichern und Laden

Befehlssyntax	Formel	Bedeutung
LDIV adr	$Akku \leftarrow M(M(adr))$	Lade einen Wert vom Speicher bei der Adresse, die bei *adr* gespeichert ist, und lade den Wert in den Akku
STIV adr	$M(M(adr)) \leftarrow Akku$	Speichere den Wert im Akku bei der Adresse, die in *adr* gespeichert ist.

MIMA Befehle: Indirektes Sichern und Laden

Befehlssyntax	Formel	Bedeutung
LDIV adr	$Akku \leftarrow M(M(adr))$	Lade einen Wert vom Speicher bei der Adresse, die bei *adr* gespeichert ist, und lade den Wert in den Akku
STIV adr	$M(M(adr)) \leftarrow Akku$	Speichere den Wert im Akku bei der Adresse, die in *adr* gespeichert ist.

Beispielprogramm mit initialem Speicherabbild

```
LDIV 4
STV 5
LDIV 5
STIV 4
HALT
```

Adresse	Wert
4	6
5	0
6	7
7	2

Lukas Bach, lukas.bach@student.kit.edu

MIMA Befehle: Eins plus Eins

Lukas Bach, lukas.bach@student.kit.edu

MIMA Befehle: Eins plus Eins

- Befehle zu arithmetischen Operationen

Lukas Bach, lukas.bach@student.kit.edu

MIMA Befehle: Eins plus Eins

- Befehle zu arithmetischen Operationen
- Eine ALU-Operation, angewandt auf dem Wert des Akkus und dem Wert an gegebener Adresse

Lukas Bach, lukas.bach@student.kit.edu

MIMA Befehle: Eins plus Eins

- Befehle zu arithmetischen Operationen
- Eine ALU-Operation, angewandt auf dem Wert des Akkus und dem Wert an gegebener Adresse
- Beispiele:

Lukas Bach, lukas.bach@student.kit.edu

MIMA Befehle: Eins plus Eins

- Befehle zu arithmetischen Operationen
- Eine ALU-Operation, angewandt auf dem Wert des Akkus und dem Wert an gegebener Adresse
- Beispiele:
  - *ADD* 4 addiert den Wert im Akku mit dem Wert aus dem Speicher an Adresse 4 und legt das Resultat im Akku ab

Lukas Bach, lukas.bach@student.kit.edu

MIMA Befehle: Eins plus Eins

- Befehle zu arithmetischen Operationen
- Eine ALU-Operation, angewandt auf dem Wert des Akkus und dem Wert an gegebener Adresse
- Beispiele:
  - *ADD* 4 addiert den Wert im Akku mit dem Wert aus dem Speicher an Adresse 4 und legt das Resultat im Akku ab. Achtung: Addition nicht mit dem Wert 4!

Lukas Bach, lukas.bach@student.kit.edu

MIMA Befehle: Eins plus Eins

- Befehle zu arithmetischen Operationen
- Eine ALU-Operation, angewandt auf dem Wert des Akkus und dem Wert an gegebener Adresse
- Beispiele:
  - *ADD* 4 addiert den Wert im Akku mit dem Wert aus dem Speicher an Adresse 4 und legt das Resultat im Akku ab. Achtung: Addition nicht mit dem Wert 4!
  - *AND* 3 führt bitweise Verundung zwischen dem Wert im Akku und dem Wert aus dem Speicher an Adresse 4 durch und legt das Resultat im Akku ab.

MIMA Befehle: Eins plus Eins

Befehlssyntax	Formel	Bedeutung
ADD adr	$Akku \leftarrow Akku + M(adr)$	Addiere den Wert bei *adr* zum Akku dazu.
"*OP*" *adr*	$Akku \text{ "}OP\text{" } M(adr)$	Wende bitweise Operation auf Akku mit Wert bei *adr* an. $Op \in \{AND, OR, XOR\}$.

MIMA Befehle: Eins plus Eins

Befehlssyntax	Formel	Bedeutung
ADD adr	$Akku \leftarrow Akku + M(adr)$	Addiere den Wert bei *adr* zum Akku dazu.
"*OP*" *adr*	$Akku \text{ "}OP\text{" } M(adr)$	Wende bitweise Operation auf Akku mit Wert bei *adr* an. $Op \in \{AND, OR, XOR\}$.

Beispielprogramm mit initialem Speicherabbild

```
LDC 5
ADD 3
AND 4
STV 5
LDC 12
XOR 5
HALT
```

Adresse	Wert
3	3
4	8
5	17

Lukas Bach, lukas.bach@student.kit.edu

MIMA Befehle: Bits und Bytes

Lukas Bach, lukas.bach@student.kit.edu

MIMA Befehle: Bits und Bytes

- *NOT* invertiert alle Bits des Datums im Akku.

Lukas Bach, lukas.bach@student.kit.edu

MIMA Befehle: Bits und Bytes

- *NOT* invertiert alle Bits des Datums im Akku. Beispiel *NOT* mit 5 im Akku, angenommen der Akku speichert bis zu 8 bits

Lukas Bach, lukas.bach@student.kit.edu

MIMA Befehle: Bits und Bytes

- *NOT* invertiert alle Bits des Datums im Akku. Beispiel *NOT* mit 5 im Akku, angenommen der Akku speichert bis zu 8 bits:
  $5_{10} = 00000101_2$, nach der Invertierung: 11111010_2.

Lukas Bach, lukas.bach@student.kit.edu

MIMA Befehle: Bits und Bytes

- *NOT* invertiert alle Bits des Datums im Akku. Beispiel *NOT* mit 5 im Akku, angenommen der Akku speichert bis zu 8 bits:
  $5_{10} = 00000101_2$, nach der Invertierung: 11111010_2.
- *RAR* rotiert alle Bits des Datums im Akku um eine Stelle nach rechts.

Lukas Bach, lukas.bach@student.kit.edu

MIMA Befehle: Bits und Bytes

- *NOT* invertiert alle Bits des Datums im Akku. Beispiel *NOT* mit 5 im Akku, angenommen der Akku speichert bis zu 8 bits: $5_{10} = 00000101_2$, nach der Invertierung: 11111010_2.
- *RAR* rotiert alle Bits des Datums im Akku um eine Stelle nach rechts. Beispiel mit 5 im Akku: 00000101_2 wird zu 10000010_2.

Lukas Bach, lukas.bach@student.kit.edu

MIMA Befehle: Bits und Bytes

- *NOT* invertiert alle Bits des Datums im Akku. Beispiel *NOT* mit 5 im Akku, angenommen der Akku speichert bis zu 8 bits: $5_{10} = 00000101_2$, nach der Invertierung: 11111010_2.
- *RAR* rotiert alle Bits des Datums im Akku um eine Stelle nach rechts. Beispiel mit 5 im Akku: 00000101_2 wird zu 10000010_2.
- *EQL adr* vergleicht den Wert im Akku mit dem Wert bei *addr*.

Lukas Bach, lukas.bach@student.kit.edu

MIMA Befehle: Bits und Bytes

- *NOT* invertiert alle Bits des Datums im Akku. Beispiel *NOT* mit 5 im Akku, angenommen der Akku speichert bis zu 8 bits: $5_{10} = 00000101_2$, nach der Invertierung: 11111010_2.
- *RAR* rotiert alle Bits des Datums im Akku um eine Stelle nach rechts. Beispiel mit 5 im Akku: 00000101_2 wird zu 10000010_2.
- *EQL adr* vergleicht den Wert im Akku mit dem Wert bei *addr*.
  - Setzt Akku $= 11 \cdots 11$ falls Werte gleich sind.

Lukas Bach, lukas.bach@student.kit.edu

MIMA Befehle: Bits und Bytes

- *NOT* invertiert alle Bits des Datums im Akku. Beispiel *NOT* mit 5 im Akku, angenommen der Akku speichert bis zu 8 bits: $5_{10} = 00000101_2$, nach der Invertierung: 11111010_2.
- *RAR* rotiert alle Bits des Datums im Akku um eine Stelle nach rechts. Beispiel mit 5 im Akku: 00000101_2 wird zu 10000010_2.
- *EQL adr* vergleicht den Wert im Akku mit dem Wert bei *addr*.
  - Setzt Akku $= 11\cdots11$ falls Werte gleich sind.
  - Setzt Akku $= 00\cdots00$ falls Werte nicht gleich sind.

Lukas Bach, lukas.bach@student.kit.edu

MIMA Befehle: Bits und Bytes

Befehlssyntax	Bedeutung
NOT	Bitweise Invertierung aller Bits des Akku-Datenwortes
RAR	Rotiere alle Akku-Bits eins nach rechts
EQL adr	Setze Akku auf $11\cdots11$, falls Wert bei *adr* gleich Akku-Wert, setze Akku auf $00\cdots00$ sonst.

Lukas Bach, lukas.bach@student.kit.edu

MIMA Befehle: Bits und Bytes

Befehlssyntax	Bedeutung
NOT	Bitweise Invertierung aller Bits des Akku-Datenwortes
RAR	Rotiere alle Akku-Bits eins nach rechts
EQL adr	Setze Akku auf $11\cdots11$, falls Wert bei *adr* gleich Akku-Wert, setze Akku auf $00\cdots00$ sonst.

Beispielprogramm mit initialem Speicherabbild

```
LDC 5
NOT
RAR
NOT
RAR
⋮
```

```
⋮
RAR
EQL 15
EQL 0
HALT
```

Lukas Bach, lukas.bach@student.kit.edu

MIMA Befehle: Springen

Lukas Bach, lukas.bach@student.kit.edu

MIMA Befehle: Springen

- Normalerweise wird die Instruktionsadresse nach jedem Befehl um eins erhöht

Lukas Bach, lukas.bach@student.kit.edu

MIMA Befehle: Springen

- Normalerweise wird die Instruktionsadresse nach jedem Befehl um eins erhöht
- Also Befehle werden von oben nach unten abgearbeitet

Lukas Bach, lukas.bach@student.kit.edu

MIMA Befehle: Springen

- Normalerweise wird die Instruktionsadresse nach jedem Befehl um eins erhöht
- Also Befehle werden von oben nach unten abgearbeitet
- Mit Sprüngen kann man die MIMA zwingen, zu definiertem Befehl zu springen und damit die Vorgehensreihenfolge zu beeinflussen

Lukas Bach, lukas.bach@student.kit.edu

MIMA Befehle: Springen

- Normalerweise wird die Instruktionsadresse nach jedem Befehl um eins erhöht
- Also Befehle werden von oben nach unten abgearbeitet
- Mit Sprüngen kann man die MIMA zwingen, zu definiertem Befehl zu springen und damit die Vorgehensreihenfolge zu beeinflussen
- *JMP adr* führt als nächsten Befehl den an Adresse *adr* aus.

Lukas Bach, lukas.bach@student.kit.edu

MIMA Befehle: Springen

- Normalerweise wird die Instruktionsadresse nach jedem Befehl um eins erhöht
- Also Befehle werden von oben nach unten abgearbeitet
- Mit Sprüngen kann man die MIMA zwingen, zu definiertem Befehl zu springen und damit die Vorgehensreihenfolge zu beeinflussen

- *JMP adr* führt als nächsten Befehl den an Adresse *adr* aus.
- *JMN adr* führt als nächsten Befehl den an Adresse *adr* aus, falls der Akku negativ ist.

Lukas Bach, lukas.bach@student.kit.edu

MIMA Befehle: Springen

- Normalerweise wird die Instruktionsadresse nach jedem Befehl um eins erhöht
- Also Befehle werden von oben nach unten abgearbeitet
- Mit Sprüngen kann man die MIMA zwingen, zu definiertem Befehl zu springen und damit die Vorgehensreihenfolge zu beeinflussen

- *JMP adr* führt als nächsten Befehl den an Adresse *adr* aus.
- *JMN adr* führt als nächsten Befehl den an Adresse *adr* aus, falls der Akku negativ ist.
  - Also wenn das erste Bit im Akku negativ ist.

MIMA Befehle: Springen

- Normalerweise wird die Instruktionsadresse nach jedem Befehl um eins erhöht
- Also Befehle werden von oben nach unten abgearbeitet
- Mit Sprüngen kann man die MIMA zwingen, zu definiertem Befehl zu springen und damit die Vorgehensreihenfolge zu beeinflussen

- *JMP adr* führt als nächsten Befehl den an Adresse *adr* aus.
- *JMN adr* führt als nächsten Befehl den an Adresse *adr* aus, falls der Akku negativ ist.
  - Also wenn das erste Bit im Akku negativ ist.
  - Wenn vorher ein *EQL* erfolgreich verglichen hat, wird also gesprungen.

Lukas Bach, lukas.bach@student.kit.edu

MIMA Befehle: Springen

- Normalerweise wird die Instruktionsadresse nach jedem Befehl um eins erhöht
- Also Befehle werden von oben nach unten abgearbeitet
- Mit Sprüngen kann man die MIMA zwingen, zu definiertem Befehl zu springen und damit die Vorgehensreihenfolge zu beeinflussen

- *JMP adr* führt als nächsten Befehl den an Adresse *adr* aus.
- *JMN adr* führt als nächsten Befehl den an Adresse *adr* aus, falls der Akku negativ ist.
  - Also wenn das erste Bit im Akku negativ ist.
  - Wenn vorher ein *EQL* erfolgreich verglichen hat, wird also gesprungen.
  - Wenn der Akku positiv ist, werden die Befehle nach *JMN* normal weiter abgearbeitet.

Lukas Bach, lukas.bach@student.kit.edu

MIMA Befehle: Springen

Befehlssyntax	Bedeutung
EQL adr	Setze Akku auf $11\cdots11$, falls Wert bei *adr* gleich Akku-Wert, setze Akku auf $00\cdots00$ sonst.
JMP adr	Springe zu Befehlsadresse *adr*
JMN adr	Springe zu Befehlsadresse *adr*, falls Akku negativ (also erstes Bit $= 1$), sonst fahre normal fort.

MIMA Befehle: Springen

Befehlssyntax	Bedeutung
EQL adr	Setze Akku auf $11\cdots11$, falls Wert bei *adr* gleich Akku-Wert, setze Akku auf $00\cdots00$ sonst.
JMP adr	Springe zu Befehlsadresse *adr*
JMN adr	Springe zu Befehlsadresse *adr*, falls Akku negativ (also erstes Bit $= 1$), sonst fahre normal fort.

Beispielprogramm mit initialem Speicherabbild

	LDC 5
a_1 :	JMN a_2
	EQL 1
	JMN a_1
	⋮
	⋮
	NOT
a_2 :	JMP a_3
	NOT
a_3 :	*HALT*

Adresse	Wert
1	5

Lukas Bach, lukas.bach@student.kit.edu

Aufgaben

MIMA-Programm schreiben

Schreibe ein MIMA-Programm:

- Eingabe: Adresse a_1 einer positiven Zahl x.
- Ausgabe: Speichert $3 \cdot x$ in a_1.

Lukas Bach, lukas.bach@student.kit.edu

Aufgaben

MIMA-Programm schreiben

Schreibe ein MIMA-Programm:

- Eingabe: Adresse a_1 einer positiven Zahl x.
- Ausgabe: Speichert $3 \cdot x$ in a_1.

Lösung:
LDV a_1
ADD a_1
ADD a_1
STV a_1
HALT

Lukas Bach, lukas.bach@student.kit.edu

Aufgaben

MIMA-Programm schreiben

Schreibe ein MIMA-Programm:

- Eingabe: Adresse a_1 einer positiven Zahl x.
- Ausgabe: Speichert x **mod** 2 in a_1.

Aufgaben

MIMA-Programm schreiben

Schreibe ein MIMA-Programm:

- Eingabe: Adresse a_1 einer positiven Zahl x.
- Ausgabe: Speichert x **mod** 2 in a_1.

Lösung:

```
LDC 1    // 000000000000000000000001
AND a1
STV a1
HALT
```

Lukas Bach, lukas.bach@student.kit.edu

Aufgaben

MIMA-Programm schreiben

Schreibe ein MIMA-Programm:

- Eingabe: Adresse a_1 einer positiven Zahl x.
- Ausgabe: Speichert x **div** 2 in a_1.

Lukas Bach, lukas.bach@student.kit.edu

Aufgaben

MIMA-Programm schreiben

Schreibe ein MIMA-Programm:

- Eingabe: Adresse a_1 einer positiven Zahl x.
- Ausgabe: Speichert x **div** 2 in a_1.

Lösung:

```
LDC 1
NOT
AND a1     // Setze “rechtestes” Bit auf 0
RAR
STV a1
HALT
```

Lukas Bach, lukas.bach@student.kit.edu