Einführung in die Informatik

- Dipl.-Inf., Dipl.-Ing. (FH) Michael Wilhelm
- Hochschule Harz
- FB Automatisierung und Informatik
- mwilhelm@hs-harz.de
- http://www.miwilhelm.de
- Raum 2.202
- Tel. 03943 / 659 338

Inhalt

Klassifizierung:

- **Einprozessorsysteme**
  - Eine CPU, John v. Neumann
- Array-Prozessorsysteme
  - Parallel arbeitende Systeme, alle Prozessoren führen die gleichen Befehle aus (Grafikkarten, OpenCL, Cuda)
- Multi-Prozessorsysteme
  - homogene MPS, gleicher CPU-Typ
  - inhomogen, heterogen MPS, unterschiedliche CPU-Typen
  - symmetrisch, jede CPU hat die identische Aufgabe
  - asymmetrisch, jede CPU hat unterschiedlichen Aufgaben
- Massiv parallele Systeme Kopplung über Netzwerke

Von-Neumann Architektur

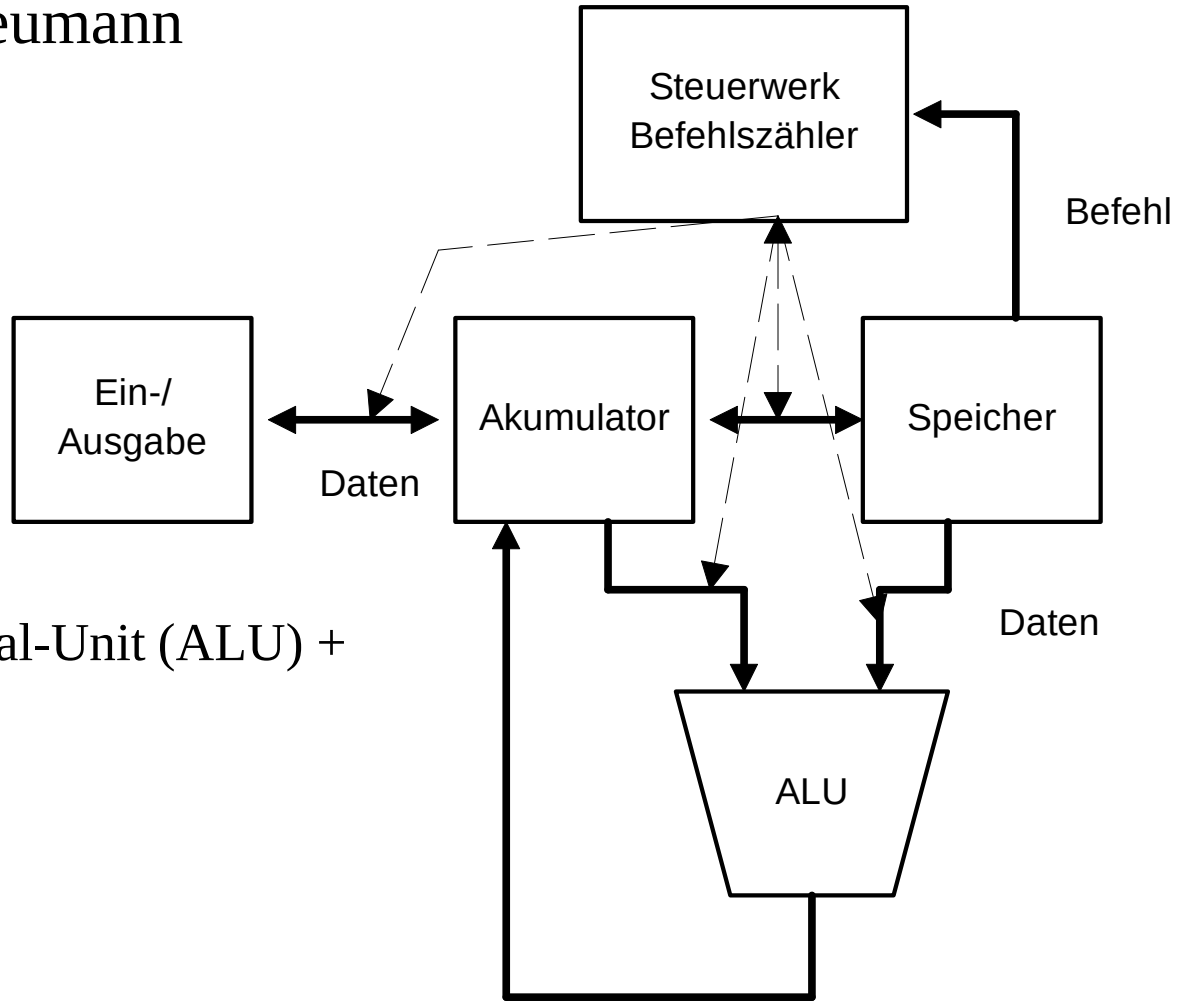
nach John von Neumann

Prozessor =

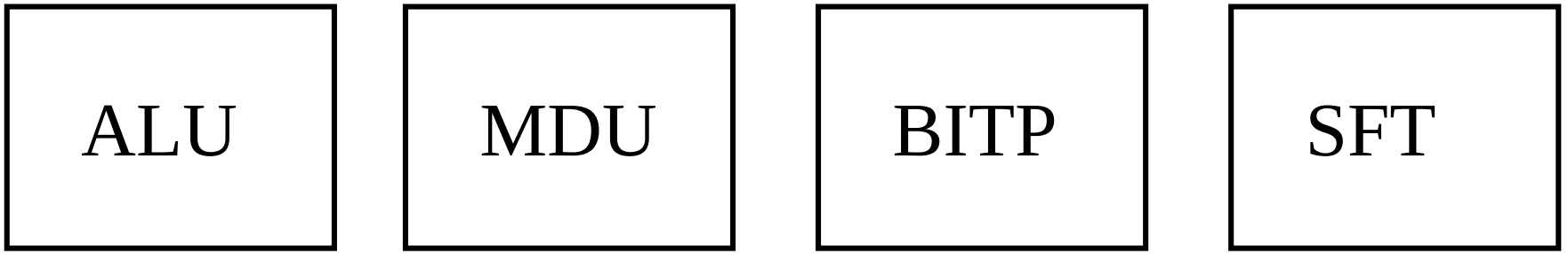
- Rechenwerk +
- Arithmetic-Logical-Unit (ALU) +
- Steuerwerk

Speicher

Module



Rechenwerk:



ALU: Arithmetisch-Logische-Einheit

MDU: Mupliplikation / Division

BITP: Bitprozessor

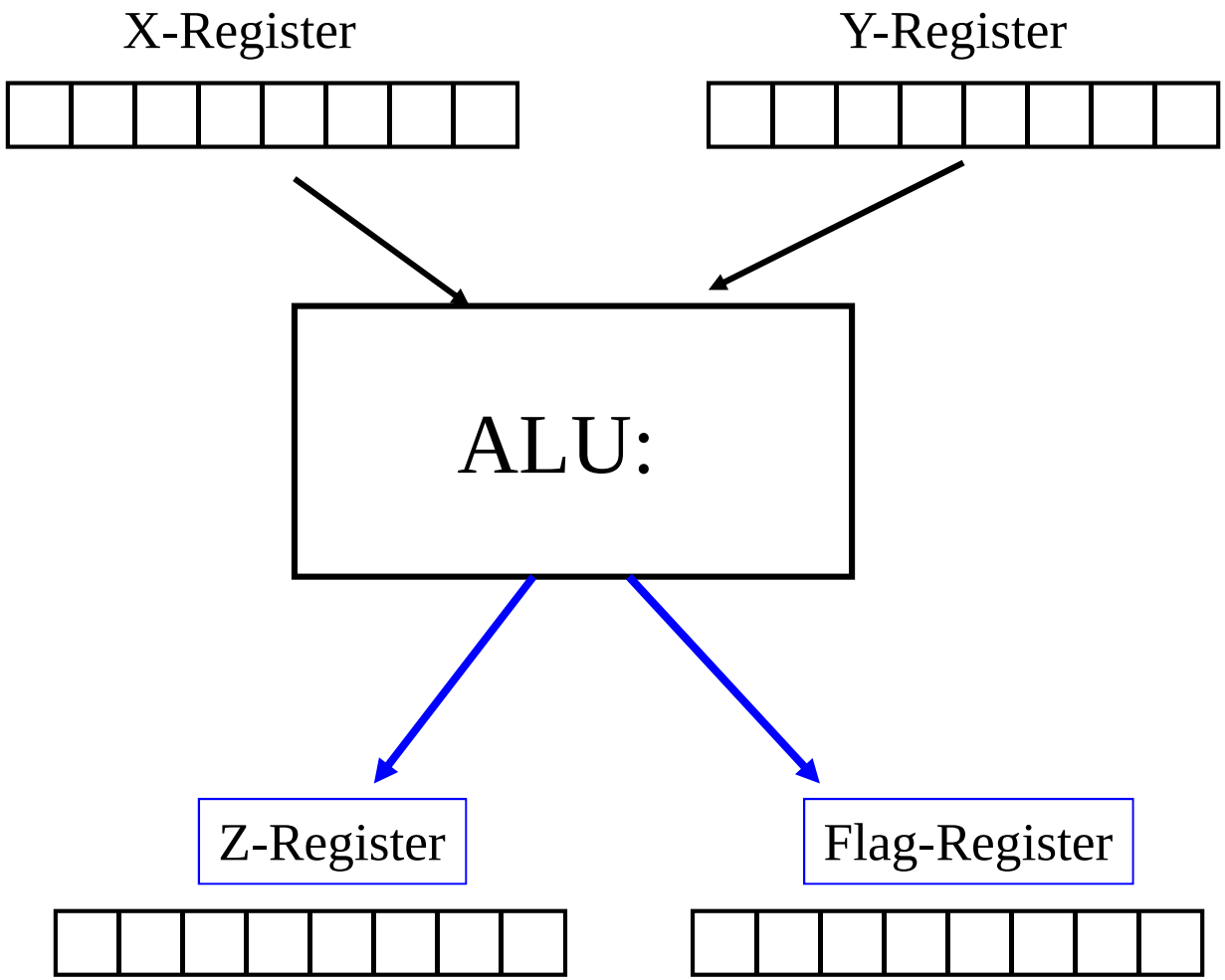
SFT: Barrel Shifter

(+) Spezialisierte Module

(+) Parallele Bearbeitung

(+) Optimierte Befehle in Zusammenarbeit mit Compiler (MMX)

Arithmetisch-Logische Einheit:



Arithmetisch-Logische Einheit:

Realisierung der Elementaroperationen:

ADD, SUB, AND, OR, XOR, NOT, EQ

Registerbreite: 8, 16, 32, 64

Ausnahmefälle:

Carry:	Ergebnis zu groß für Register Z (Überlauf)
Overflow:	Ergebnis zu groß für die Zahlendarstellung
Underflow:	Ergebnis zu klein für die Zahlendarstellung
Sign:	Ergebnis ist negativ
Zero:	Ergebnis ist null

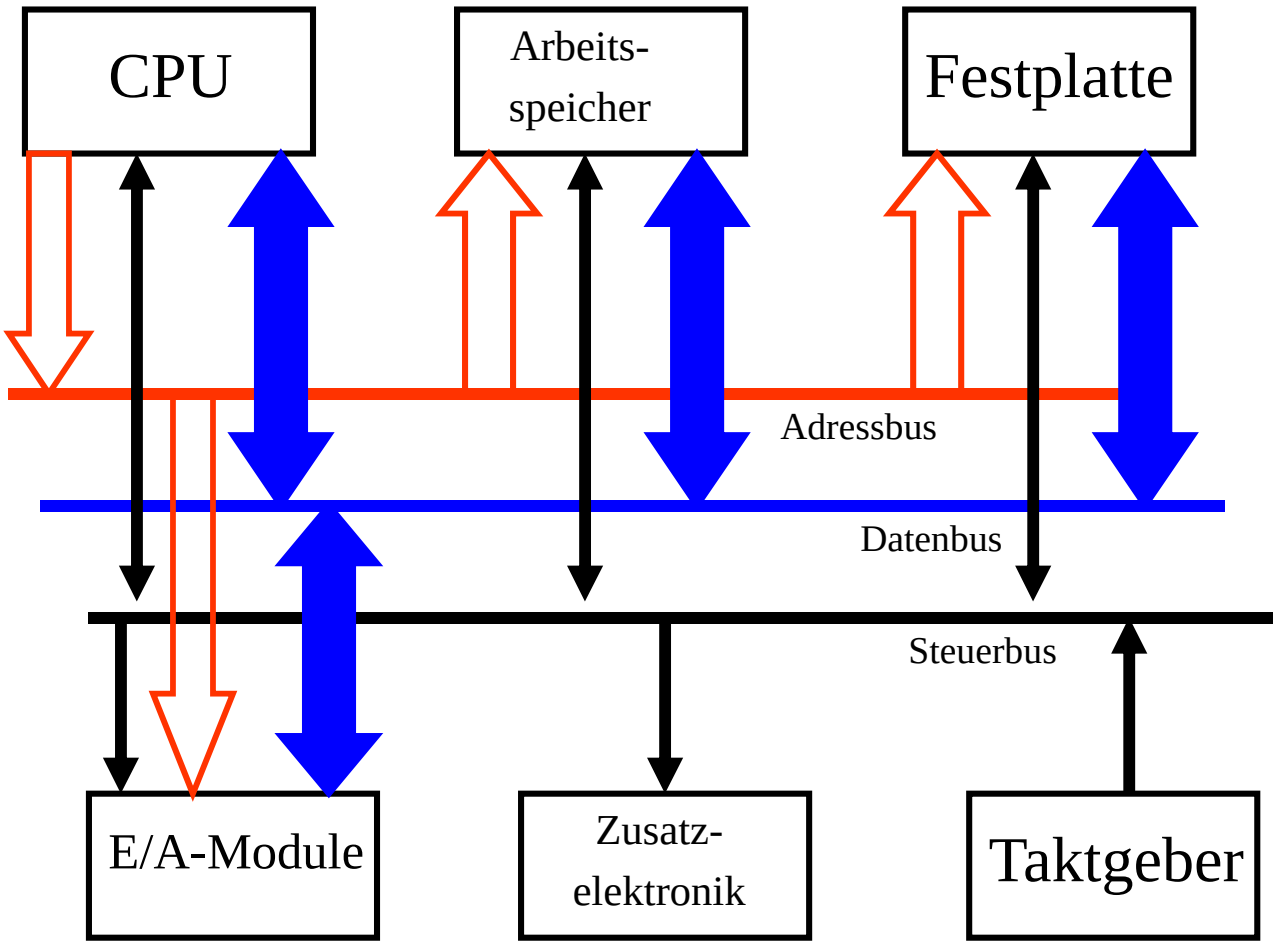
- Operationsprinzipien
  - Der Prozessor (die Central Procesing Unit, CPU) arbeitet taktgesteuert.
  - Die internen Signale sind binär codiert.
  - Der Inhalt einer Speicherzelle wird über eine Adresse angesprochen.
  - Programme und Daten werden sequentiell bearbeitet (single instruction single data, SISD).
  - Es wird eine feste Wortlänge zur Verarbeitung verwendet.
- Bitmuster im Speicher repräsentieren 3 Arten:
  - Daten, Befehle, Adressen

Die Elemente des Prozessors

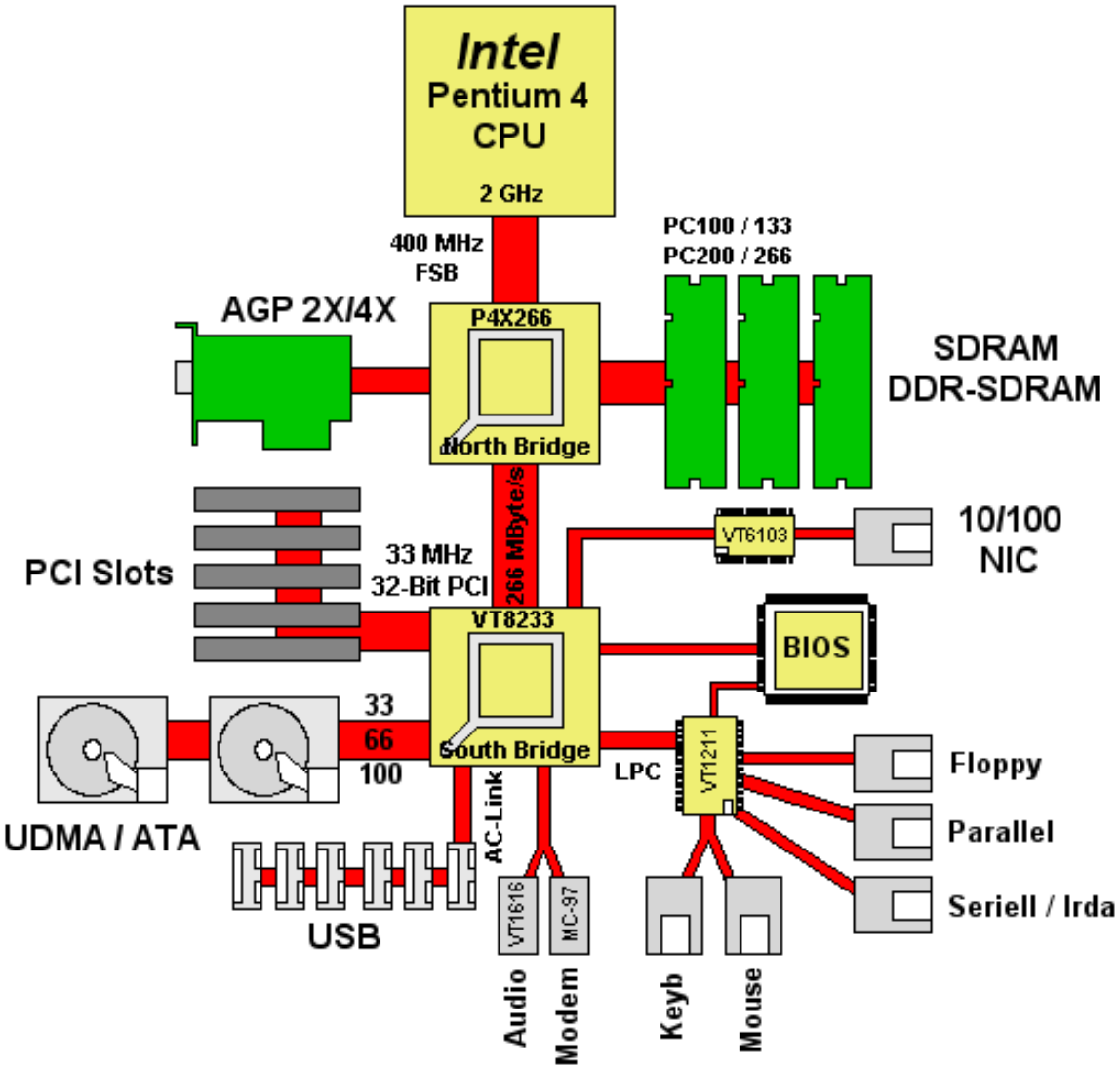
- Das **Steuerwerk** enthält den Befehlszähler (Adresse des nächsten Befehls), das Befehlsregister und das Adressregister (Adresse des nächsten Datums)
- Das **Rechenwerk** (ALU) enthält mehrere Datenregister und Zusatzregister (Flagregister). Die Verknüpfung erfolgt hier gleichzeitig innerhalb eines Takts mit allen Bits je nach Datenbreite
- Einsatz von **Mikrocode** zur Steuerung der internen Abläufe: eigener Speicher (ROM), Adressen der Register, Steuerwerk zur Abfolge von Mikrobefehlen in mehreren Takten, Ziel: maximale Parallelisierung bei minimalem Aufwand an Logikschaltungen

Modifizierte Architektur

- Busse
  - Adressbus
  - Datenbus
  - Steuerbus
- Mikro-prozessor



Motherboard: PCI-BUS



Quelle: http://www.hardwaregrundlagen.de)

Aufgaben eines Motherboards

- Die Northbridge sollte die Daten so schnell transportieren können wie sie die CPU liefert.
- Der Arbeitsspeicher sollte die Daten so schnell aufnehmen können wie sie von der CPU kommen.
- Die Interne Verbindung (Datenautobahn) zwischen North- und Southbridge sollte so groß sein wie nur möglich, um einen Flaschenhals zu vermeiden. Hier krankt es bei den meisten Mainboards erheblich.

Quelle: http://www.hardwaregrundlagen.de)

Logische Verknüpfungen

ODER	0	1
0	0	1
1	1	1

UND	0	1
0	0	0
1	0	1

Logische Verknüpfungen

	0	**1**
NOT	1	0

XOR	**0**	**1**
0	0	1
1	1	0

XOR ergibt eine 1, wenn die Bits unterschiedlich sind!

Logische Verknüpfung NAND

NAND (a,b) = Not And (a,b)

$$c = \overline{a \wedge b} = \overline{a} \vee \overline{b}$$

a	b	c
0	0	1
0	1	1
1	0	1
1	1	0

Logische Verknüpfung NOR

NOR (a,b) = Not Or (a,b)

$$c = \overline{a \vee b} = \overline{a} \wedge \overline{b}$$

a	b	c
0	0	0
0	1	0
1	0	0
1	1	1

Beispiele:

Diese logische Verknüpfungen werden auch auf Bitfolgen angewandt.

0001	ODER	1000	=	1001
0001	OR	1000	=	1001
0001	$\vee$	1000	=	1001
0001	|	1000	=	1001

1011	UND	1000	=	1000
1011	AND	1000	=	1000
1011	$\wedge$	1000	=	1000
1011	&	1000	=	1000

Beispiele:

Diese logische Verknüpfungen werden auch auf Bitfolgen angewandt.

NOT	1010	=	0101
NICHT	1010	=	0101
¬	1010	=	0101
!	1010	=	0101

1011	XOR	1000	=	0011
1010	XOR	1111	=	????
0101	XOR	1111	=	????

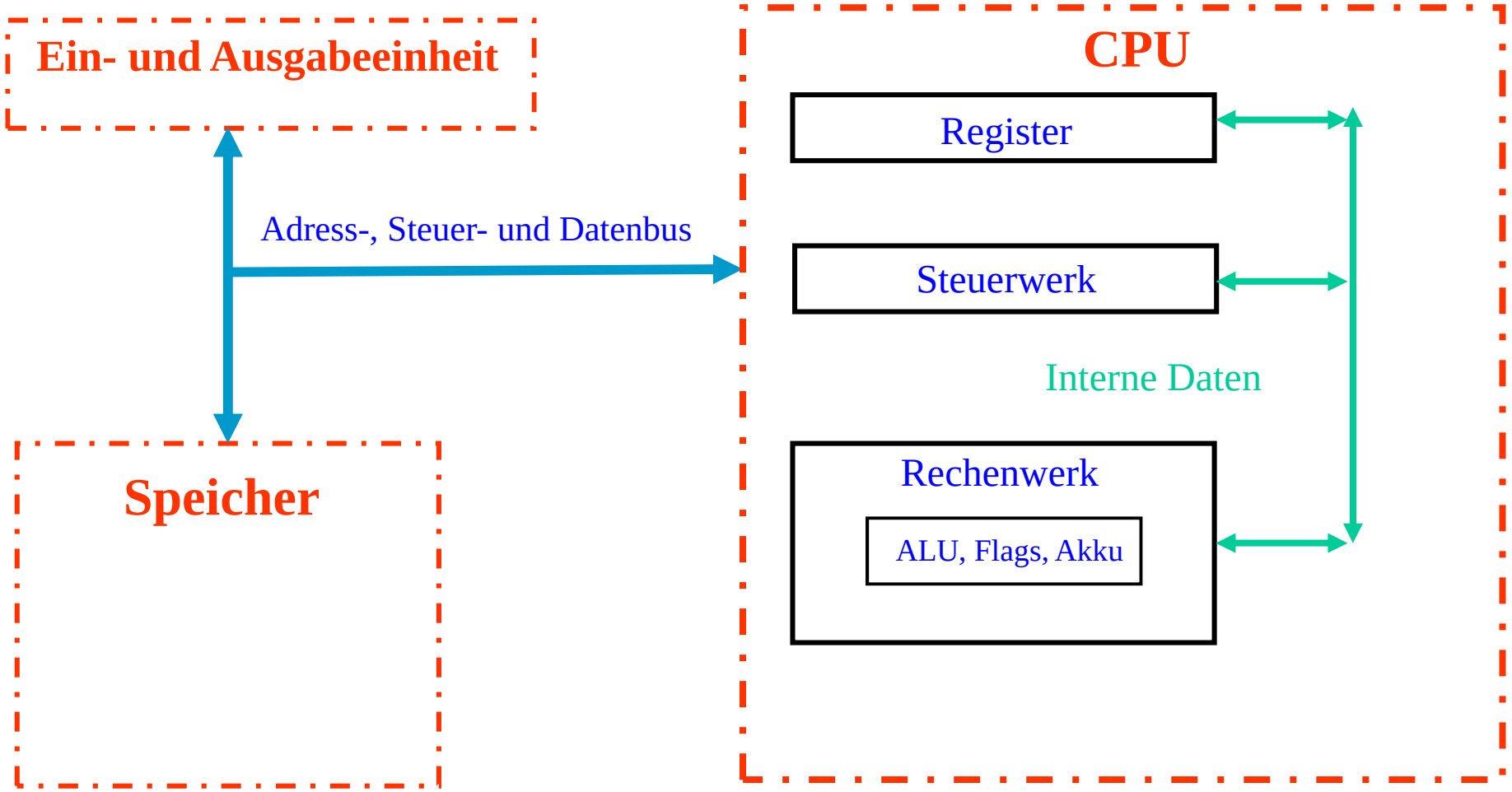
Kommunikationsgesetz

Das Kommunikative Gesetz erlaubt das Vertauschen der in einer UND bzw. ODER-Verknüpfung verwendeten Variablen

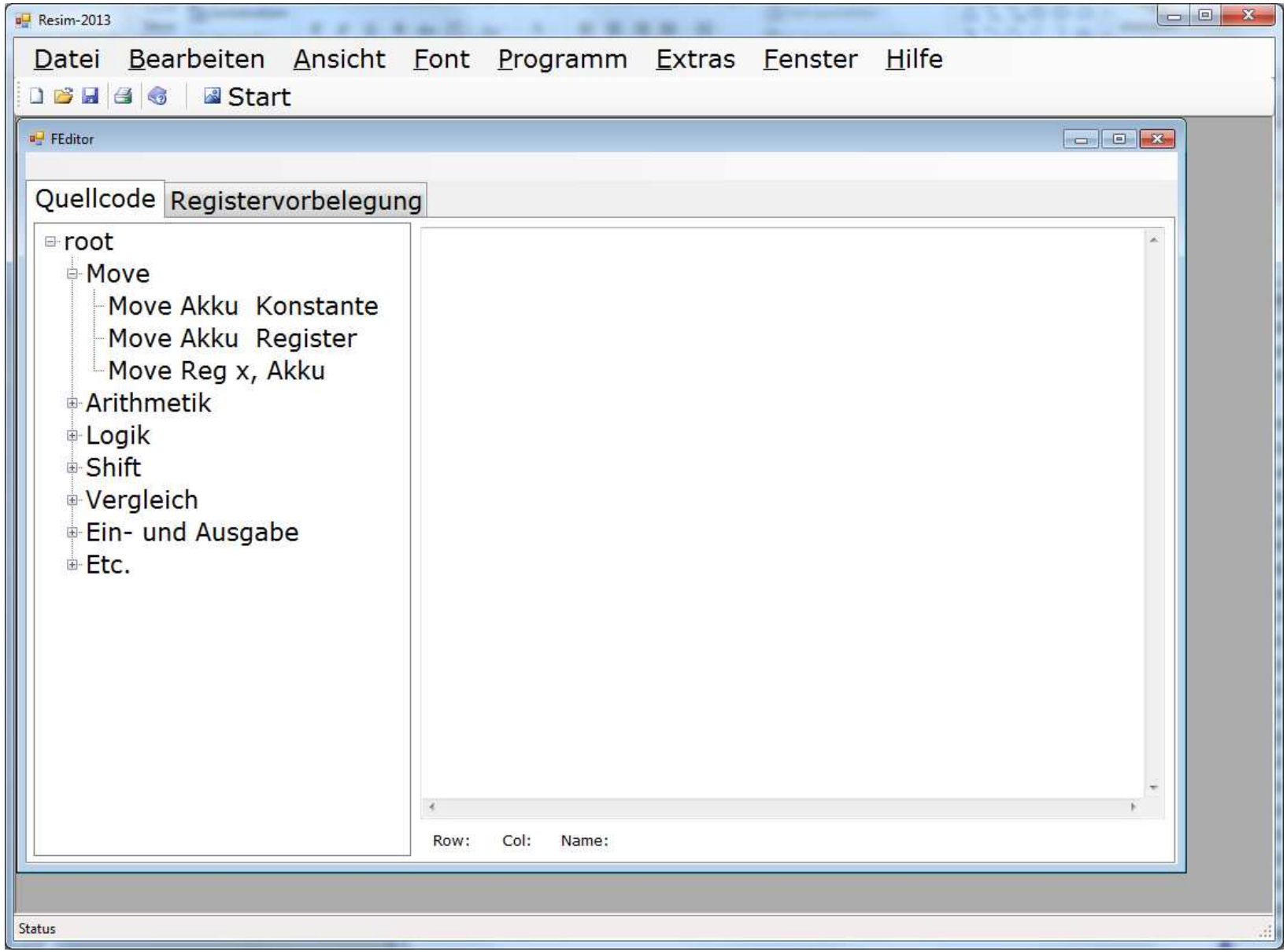
$$a \wedge b \wedge c = b \wedge c \wedge a$$

$$a \vee b \vee c = b \vee c \vee a$$

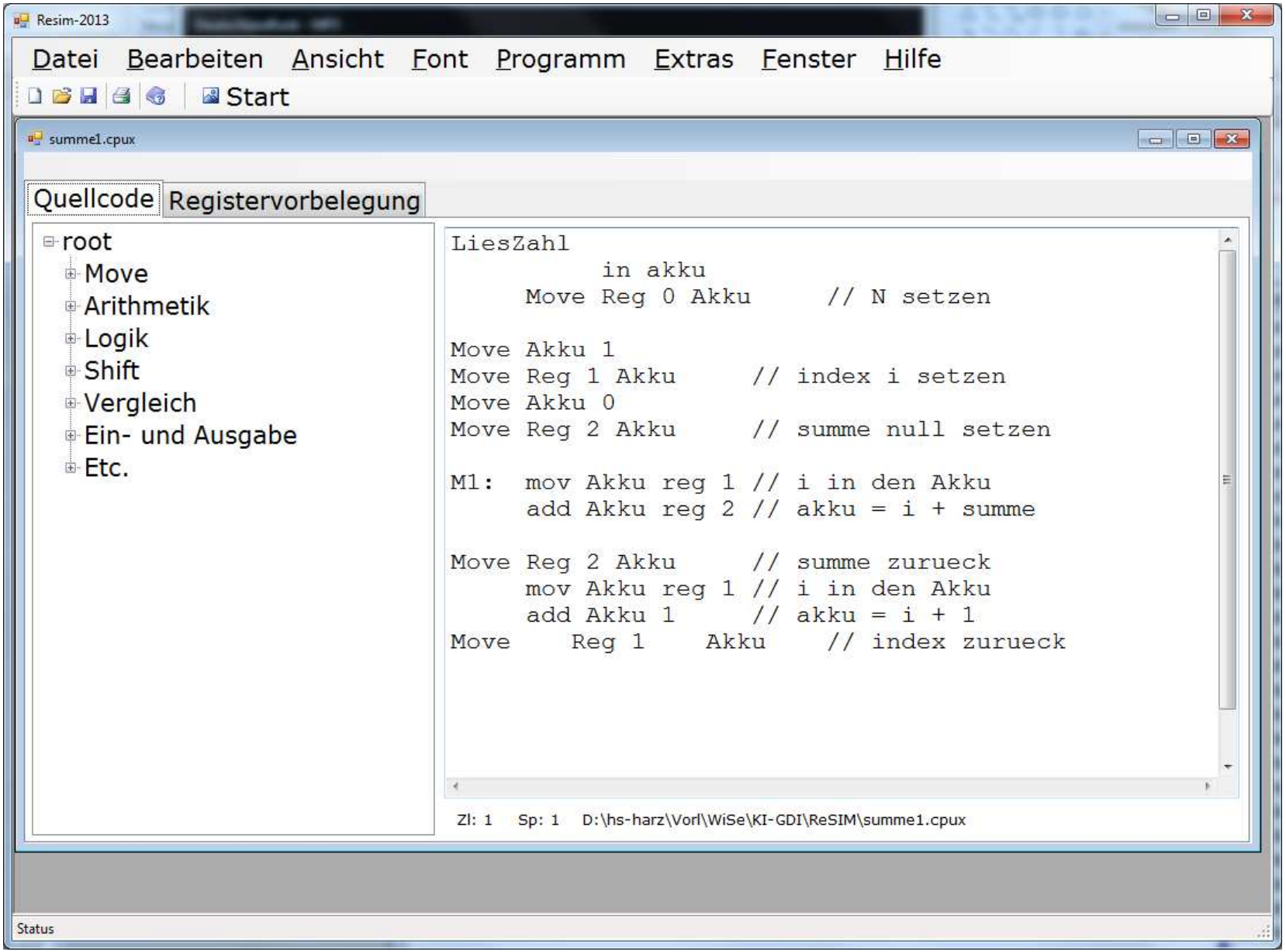
Rechnersimulation ReSim: Blockdiagramm



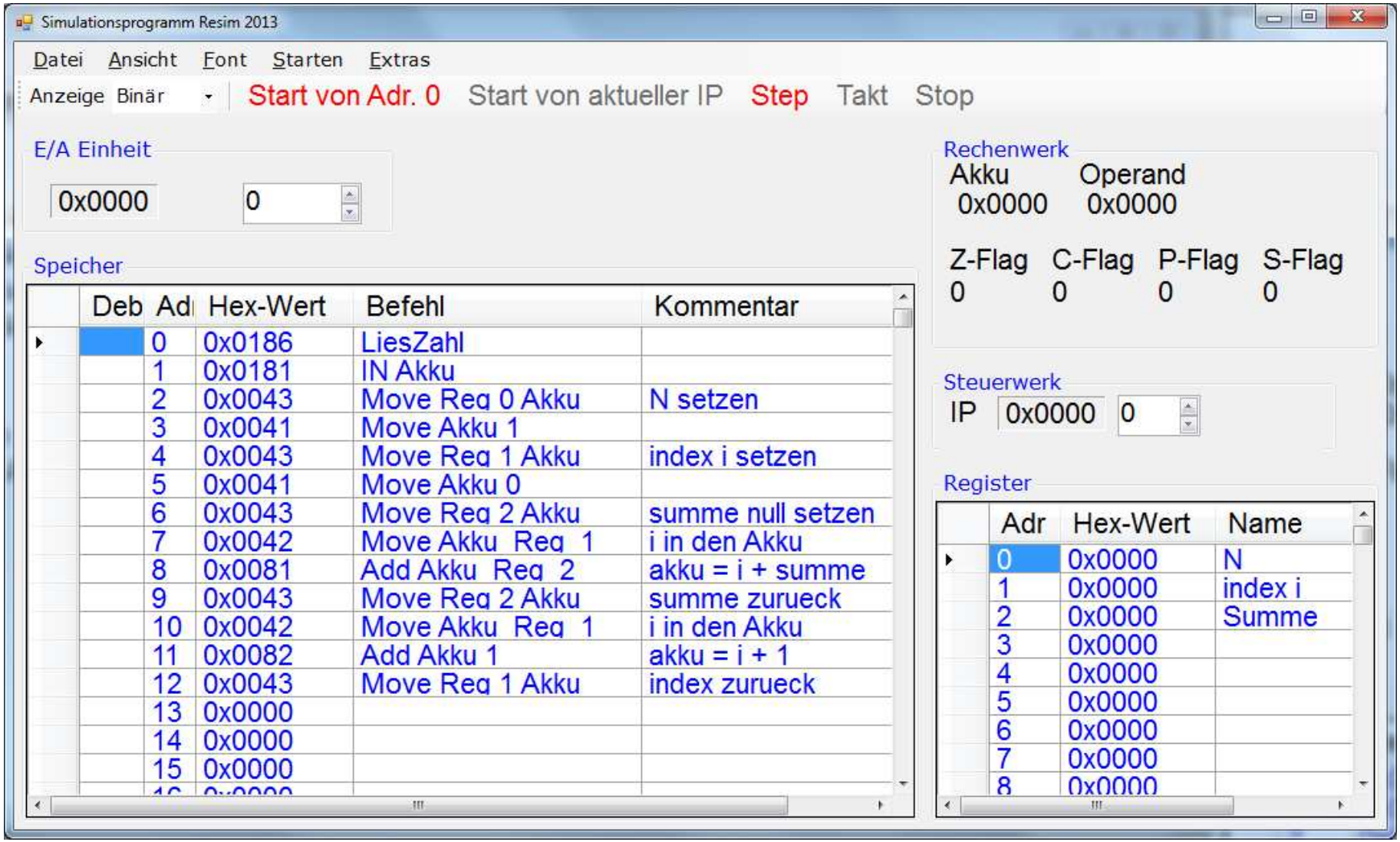
Rechnersimulation ReSim



Rechnersimulation ReSim



Rechnersimulation ReSim



ReSim: Befehle

- Transportbefehle
  - MOV Akku {Konstante, Register}
  - MOV {Register}, Akkumulator
- Arithmetikbefehle
  - ADD Akku {Konstante, Register}
  - SUB Akku {Konstante, Register}
  - MOD Akku {Konstante, Register}
  - MULT Akku {Konstante, Register}
  - DIV Akku {Konstante, Register}
  - INC Akku
  - DEC Akku
- Logik-Befehle
  - AND Akku {Konstante, Register}
  - OR Akku {Konstante, Register}
  - XOR Akku {Konstante, Register}
  - NOT Akku
- Schiebe-Befehle
  - SHL, Akku {Konstante, Register}
  - SHR, Akku {Konstante, Register}
  - ROL, Akku Konstante
  - ROR, Akku Konstante

- Sprungbefehle
  - CMP , {Konstante, Register}
  - JUMP Adresse
  - JZ: nach Adresse wenn Zero-Flag gesetzt
  - JNZ: nach Adresse wenn Zero-Flag nicht gesetzt
  - JS: nach Adresse wenn Sign-Flag gesetzt
  - JNS: nach Adresse wenn Sign-Flag nicht gesetzt
  - JG: nach Adresse wenn Akku>0
  - JGE: nach Adresse wenn Akku >= 0
  - JL: nach Adresse wenn Akku <0
  - JLE: nach Adresse wenn Akku <=0
  - JC: nach Adresse wenn Carry-Flag gesetzt
  - JNC: nach Adresse wenn Carry-Flag nicht gesetzt
  - JP: nach Adresse wenn Parity-Flag gesetzt
  - JNP: nach Adresse wenn Parity-Flag nicht gesetzt
- E/A-Befehle
  - LiesZahl „Eingabe von N:"
  - IN {Akku Register}
  - OUT {Akku Register}
  - OUT NL // NewLine
  - OUT String „Hier ist ein Text"
- Steuerbefehle
  - Clear Carry
  - Goto Sub / IRET
  - NOP (No Operation)

Statusflags: Änderung nach einer Operation

- Zero-Flag
  - Das Zero-Flag wird immer gesetzt, wenn der Akku den Wert Null hat
- Sign-Flag
  - Das Sign-Flag wird bei einem negativem Akku-Wert gesetzt
- Carry-Flag
  - Das Carry-Flag wird bei einem Überlauf einer Operation gesetzt (65535+1)
- Parity-Flag
  - Das Parity-Flag wird gesetzt, wenn die Summe der gesetzten Bits gerade ist
- Overflow-Flag
  - Das Overflow-Flag wird gesetzt, wenn der neue Wert nicht in das Ergebnisformat passt (Sign-Int bei Division)
- IP
  - Instruktion-Pointer (Befehlszähler)
- SP
  - Stackpointer. Zeigt auf einen zusätzlichen Speicher des aktuellen Programms

1. Beispiel

Addition einer Zahl um 4:

Eine Zahl ist im Register 0.
Sie soll um 4 erhöht werden.

Lösung:

Eine direkte Addition im Register / Speicher ist unmöglich !!

Ablauf:

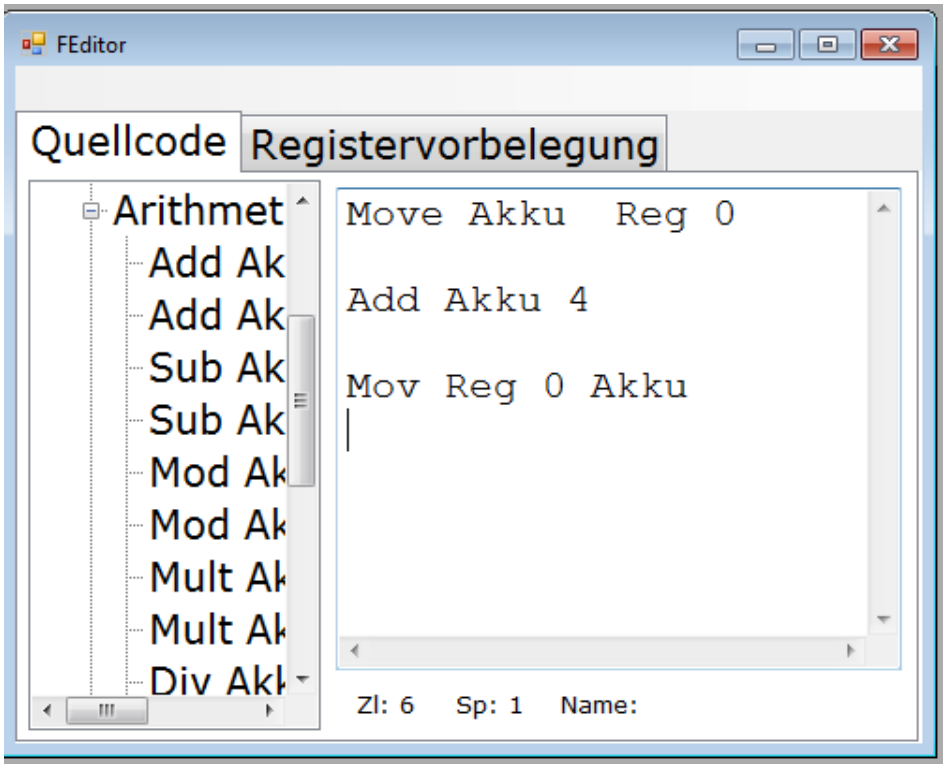
- Kopieren aus dem Register in den Akku
- Addieren um 4
- Kopieren in den Register

1. Beispiel: Lösung

Voraussetzungen:

In Register steht die Zahl 3

Reg0 := 3



Ablauf programmieren:

- Kopieren aus dem Register in den Akku
  - Mov Akku Reg 0
- Addieren um 4
  - Add Akku 4
- Kopieren in das Register
  - Mov Reg 0 Akku

Programm starten:

- Taste F5
- Einzelschritt

2. Beispiel

Die Zahl 155 soll auf das Bit 2 getestet werden:

Lösung:

Bit 2 hat den Wert: 4

Ablauf:

- Zahl 155 in den Akku kopieren
- Boolesche Operation mit der Zahl 4
- Kopieren in das Register 1

2a. Beispiel

Die Zahl 155 soll auf das Bit 2 und 5 getestet werden:

Lösung:

Bit 2 hat den Wert: 4

Bit 5 hat den Wert: 32

Ablauf:

- Zahl 155 in den Akku kopieren
- Boolesche Operation mit der Zahl 4
- Kopieren in das Register 1

2b. Beispiel

In der eingelesene Zahl soll das dritte Bit gesetzt werden

3. Beispiel

Die Zahl 155 soll auf die Bits 0, 2, 5 getestet werden:

Lösung:

Bits 0, 2, 5 haben zusammen den Wert: (1+4+32)

Ablauf:

- Zahl 155 in den Akku kopieren
- Boolesche Operation mit der Zahl ???
- Kopieren in das Register 1

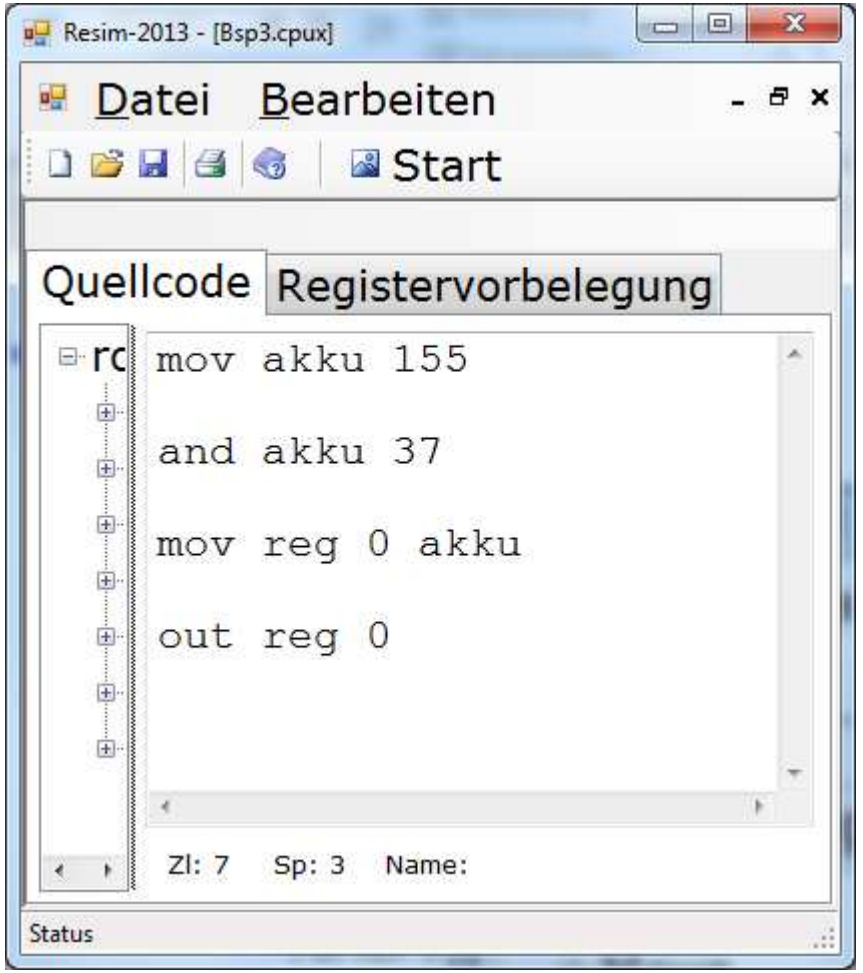
3. Beispiel: Lösung

Ablauf programmieren:

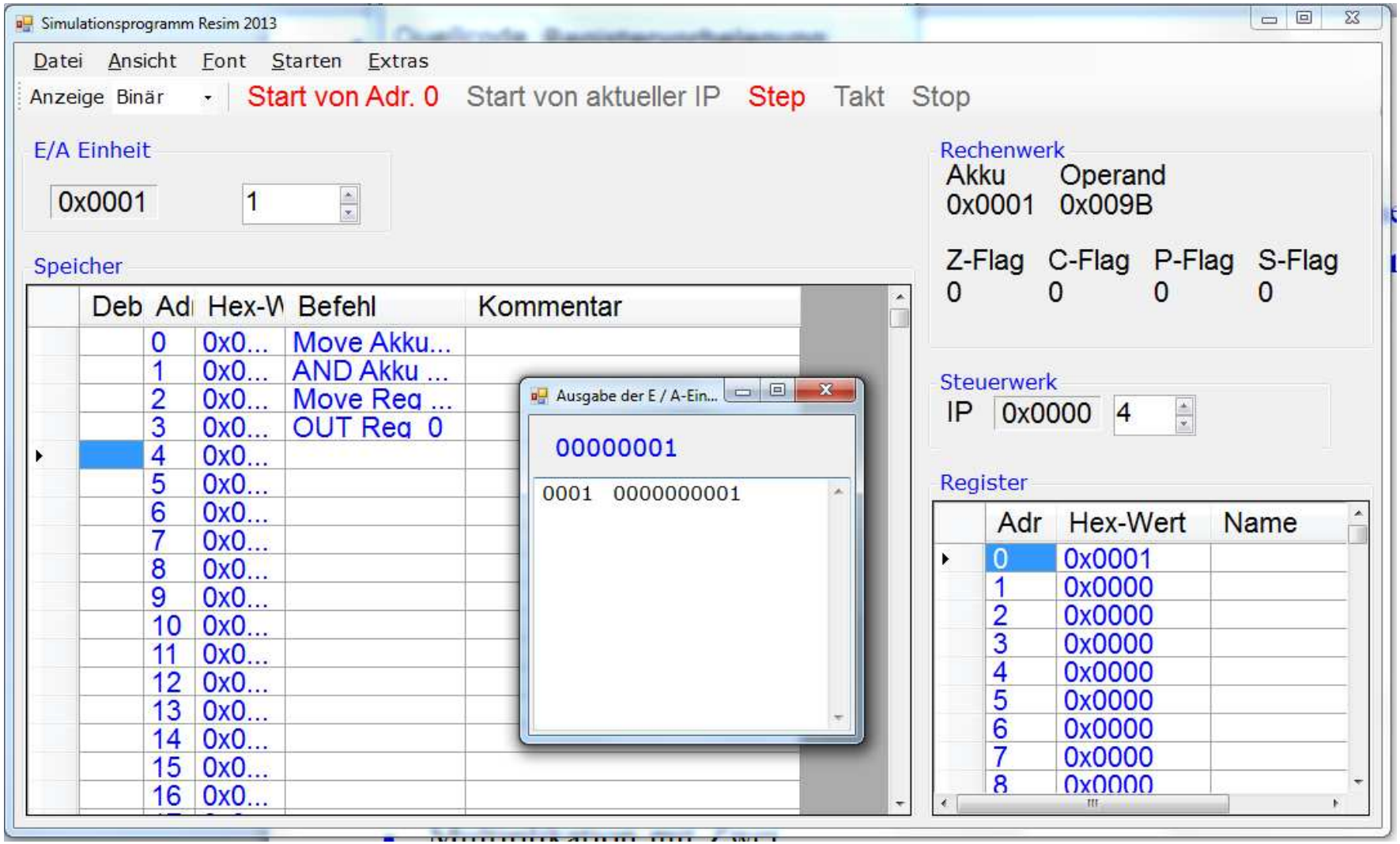
- Akku füllen mit 155 (Konstante)
  - Mov Akku 155
- Bool´sche Operation AND mit Konstante
  - AND Akku 37
- Kopieren in das Register
  - Mov Reg 0 Akku
- Ausgabe in der E/A-Einheit
  - out reg 0

Programm starten:

- Taste F5
- Einzelschritt mit Schalter „Step“



3. Beispiel im Debugger



4. Beispiel

Es soll durch die E/A-Einheit eine Zahl eingelesen werden, diese ins Register Null kopiert werden und dann dieser Inhalt möglichst schnell mit zwei multipliziert werden. Ausgabe in Register 1

Lösung:

Ablauf:

- Zahl durch E/A-Lesen in die E/A-Einheit einlesen
- Kopieren ins Register Null
- Multiplikation nur im Akku: mov, Reg0 ins Akku
- Multiplikation mit Zwei
- Ausgabe ins Register 1

4. Beispiel: Lösung

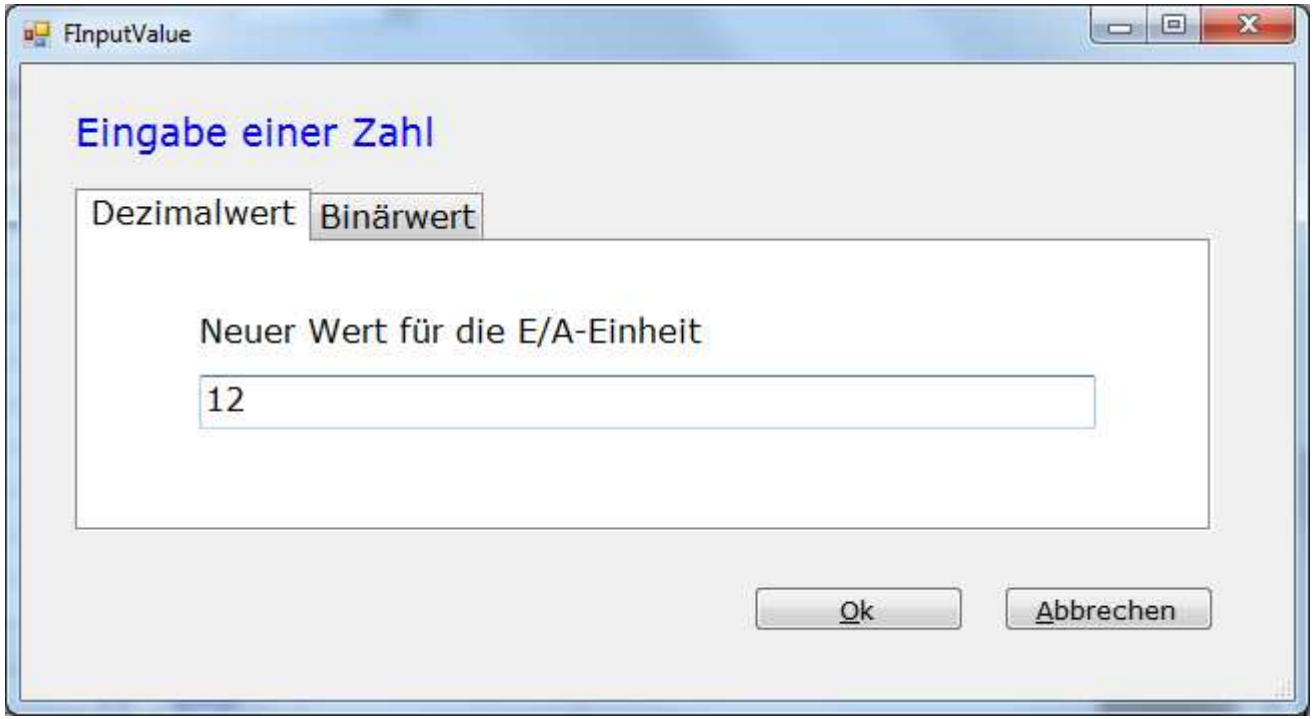
Ablauf programmieren:

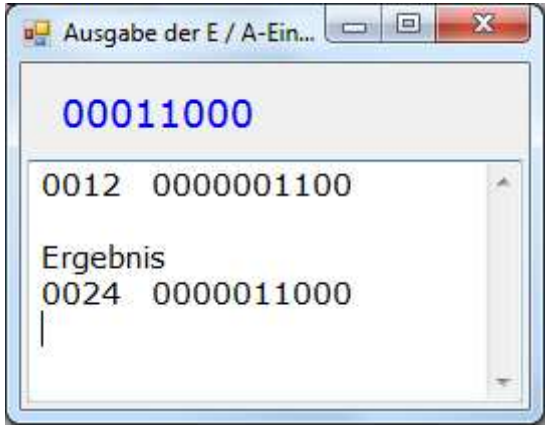
- Lese ein Zahl mittels Eingabedialog in die E/A-Einheit
  - LiesZahl "Zahl x"
- Kopieren in Register 0
  - In Reg 0
- Kopieren in den Akku
  - Mov Akku Reg 0
- Multiplikation mit Shift Links
  - SHL akku 1
- Ergebnis ins Register 1
  - Mov Reg 1 Akku
- Ergebnis ins Register 1
  - out String "Ergebnis"
  - out akku

Programm starten:

- Taste F5
- Einzelschritt

4. Beispiel: Debugger





Berechnung von a-b

Registerbelegung:

- Reg0: Zahl a
- Reg1: Zahl b
- Reg2: 2-er Komplement von b
- Reg3: a-b
- Reg4: positive Zahl ermitteln, falls das Ergebnis negativ ist (Kontrolle)

Berechnung der Quersumme

Registerbelegung:

- Reg0: Zahl x
- Reg1: Laufindex
- Reg2: Summe

Ablauf:

- Einlesen der Zahl x
- Speichern in Reg0
- Weitere Initialisierungen
- Schleife
- Ausgabe der Summe