Handbuch zum Rechner-Simulationsprogramm

ReSim 2015

Version 7.02

HS Harz
Fachbereich
Automatisierung und Informatik

von

Dipl. Inf., Dipl.-Ing. (FH) Michael Wilhelm

http://www.miwilhelm.de

mwilhelm@hs-harz.de

Version 26. Oktober 2015

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einführung

Das vorliegende Programm dient dazu, die John-von-Neumann-Architektur an Hand von Beispielen näher zu erläutern. Dieses Programm wurde als erstes von Herr Meißner als Abschlussarbeit entwickelt. Danach als nach MFC und Delphi portiert. Die aktuelle Fassung wurde mit Winforms und C# programmiert und hat etliche Erweiterungen erfahren. Eine Java-Version ist auch auf meiner Homepage unter der Rubrik „Download/Resim“ zu finden.

1.1 John von Neumann

Die klassische von-Neumann-Architektur wurde im Jahre 1946 von John von Neumann als Konzept zur Gestaltung eines universellen Rechners für technische, wissenschaftliche und wirtschaftliche Problemstellungen vorgeschlagen. Bis auf wenige Ausnahmen sind die heutigen Rechenanlagen Weiterentwicklungen dieses Universalrechners.

Nach [1] ist ein Rechner nach in der von-Neumann-Architektur aufgebaut [1], wenn er folgende Kriterien erfüllt:

- Der Rechner besteht aus fünf Komponenten,
  - dem Steuerwerk
  - dem Rechenwerk
  - dem Speicher
  - dem Eingabewerk
  - dem Ausgabewerk
- Die Struktur des Rechners ist unabhängig von der Art der Problemstellung. Dem Rechner muss ein Programm als Bearbeitungsvorschrift vorliegen, ansonsten ist er arbeitsunfähig
- Programm und Daten sind in einem Speicher untergebracht
- Der Speicher ist in gleich große, durchnummerierte Zellen (Speicherzellen) unterteilt.
- Aufeinanderfolgende Programmbefehle werden in aufeinander folgenden Speicherzellen gespeichert
- Durch Sprungbefehle kann die Bearbeitung dieser Reihenfolge verlassen werden
- Es gibt mindestens folgende Befehlsarten:
  - Arithmetikbefehle
  - Logikbefehle
  - Schiebeoperationen
  - Transportbefehle
  - bedingte und unbedingte Sprünge
  - Befehle für Unterbrechungen
  - Wartebefehle
  - Ein- /Ausgabebefehle
- Alle diese Befehle können in verschiedenen Adressierungsarten ausgeführt werden.
- Alle Befehle und Daten sind binär kodiert, so dass eine Unterscheidung nicht möglich ist. Geeignete Schaltwerke im Steuerwerk decodieren die Daten und sorgen für die richtige Entschlüsselung.

Die Rechnerarchitektur des von-Neumann-Rechners bezeichnet man nach M. Flynn als SISD-Architektur[1]. Das bedeutet, dass immer nur ein Befehl auf ein Datum, ein Zahlenwert, angewandt werden kann. Im Gegensatz dazu gibt es Architekturen, die mehrere Befehle auf ein Datum anwenden (MISD), die einen Befehl auf mehrere Daten anwenden (SIMD) und Architekturen, die mehrere Befehle gleichzeitig auf mehrere Daten anwenden können (MIMD). Die Tätigkeit eines von-Neumann-Rechners besteht im Wesentlichen im Transportieren von Daten zwischen dem Rechenwerk und dem Speicher bzw. Ein-/Ausgabewerk. Dies geschieht alles auf nur einem Datenkanal. Aus diesem Grunde nennt man diesen Pfad auch den "von-Neumann-Flaschenhals". Hier können die oben genannten Architekturen Abhilfe schaffen. Die heutigen Rechnerarchitekturen benutzen mehrere unabhängige Bus-Systeme, zum Beispiel den PCI-Bus mit der North- und South-Bridge.

Die Abbildung 1 zeigt den prinzipiellen Aufbau eines John-von-Neumann-Rechners. Die Befehle, Daten und Adressen sind im Speicher.

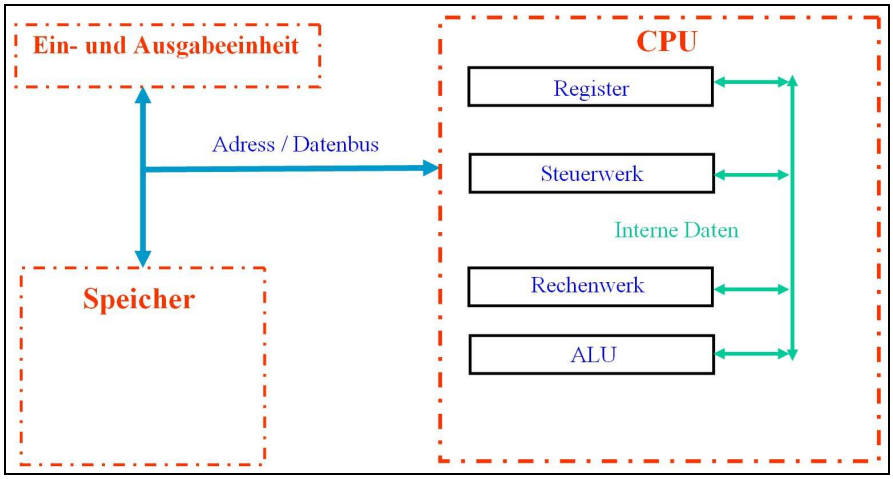


Abbildung 1 Prinzip-Skizze eines John-von-Neumann-Rechners

Die Daten und Programme werden durch das Betriebssystem von Festplatte in den Hauptspeicher geladen. Prinzipiell besteht der Speicher nur aus Nullen und Einsen. Welche Bytes Daten bzw. Programmbefehle sind, kann man ohne Analyse **nicht** entscheiden. Der zweite wichtige Aspekt ist das Steuerwerk. Dieses liest die Befehle aus dem Speicher. Dazu benutzt es den Adressbus zur Auswahl des Befehls bzw. der Daten. Im zweiten Schritt werden über dem Datenbus die Datenbits in das Steuerwerk transportiert. Dort wird der Befehle analysiert, und es werden gegebenenfalls noch weitere Operanden aus dem Speicher geholt. Danach werden die Daten in das Rechenwerk bzw. der ALU[2] transportiert und der Befehl ausgeführt. Das Ergebnis wird dann in Register oder in einem Speicher zurückkopiert. Nach Abarbeitung des Befehls wird der nächste Befehl zur Ausführung geholt.

1.2 Steuerwerk

Das Steuerwerk ist für die Zerlegung der Befehle, der Dekodierung der Daten und die Generierung der Steuersignale zuständig. Von hier aus werden die Speicherzellen und die Register adressiert sowie die Schreib- und Leseleitungen aktiviert.

[1] SISD: Single Instruction, Single Data
[2] ALU = Arithmetic Logical Unit

Zuerst werden die Befehle in Mikrocode Befehle, auch Nanobefehle genannt, zerlegt, danach werden die einzelnen Steuerleitungen geschaltet. Hier werden die Bitkombinationen der Befehle in elektrische Impulse umgewandelt.

1.3 Rechenwerk

Das Rechenwerk besteht aus den beiden Operandenregistern und dem AKKU als Zielregister. Außerdem gibt es noch ein Übertrag-Flag und die ALU, in der die eigentliche Verknüpfung der Daten geschieht. Die "Programmierung" der ALU geschieht über den Operandenbus OP.
Der Datenspeicher besteht aus acht Speicherzellen und dient der Speicherung von Daten, die nicht unmittelbar verarbeitet werden sollen. Wenn das geschehen soll, müssen die Daten erst in den AKKU oder in eines der acht Register transportiert werden. Die Zugriffszeiten heutiger Datenspeicher liegen zwischen 1 bis 100 ns. Die Speichergröße kann bis zu einigen GByte betragen.

1.4 Ein- und Ausgabeeinheit

Die Ein-/Ausgabeeinheit besteht aus zwei Daten-Kanälen, im Programm aus 8-Bit, wobei einer als Eingabekanal und einer als Ausgabekanal dient. Hier können Daten eingelesen oder ausgegeben werden. Aktuell werden die Daten in einem Dialogfenster eingegeben.

1.5 Register

Die Registerbank besteht aus acht Registern und dient der Aufnahme von Operanden und Zwischenergebnissen. Von hier aus können Daten direkt mit dem AKKU verknüpft werden. Die Zugriffszeiten sind aufgrund der einfacheren Adressierung und anderem hardwaremäßigen Aufbau bedeutend kürzer als bei Datenspeichern. Die heutigen Prozessoren haben zwischen 16 und 128 Register mit bis zu 128 Bit Datenbreite.

1.6 Datenbus

Über den Datenbus werden die Daten zwischen den einzelnen Komponenten transportiert. Er verbindet den Speicher, das Rechenwerk, die Ein-/Ausgabeeinheit und die Registerbank miteinander.

1.7 Adressbus

Über den Adressbus werden die einzelnen Speicherzellen und Register angesprochen. Die Adresse wird vom Steuerwerk generiert und auf den Adressbus ausgegeben. Jede Speicherzelle und jedes Register ist mit dieser Adresse eindeutig identifizierbar. Die Steuerleitungen wie z.B. IORD oder WR legen fest, ob eine Information gelesen oder geschrieben werden soll. Sie werden ebenfalls vom Steuerwerk generiert.

2 Aufbau des Programms

Die Oberfläche ist als MDI-Programm aufgebaut. Es können beliebig viele Resim-Quelldateien in jeweils einem Editor bearbeitet werden. Mit dem Befehl „Starten“ wird in das CPU-Fenster gewechselt. Dieses Fenster kann nur ein Programm bearbeiten.

2.1 Oberfläche

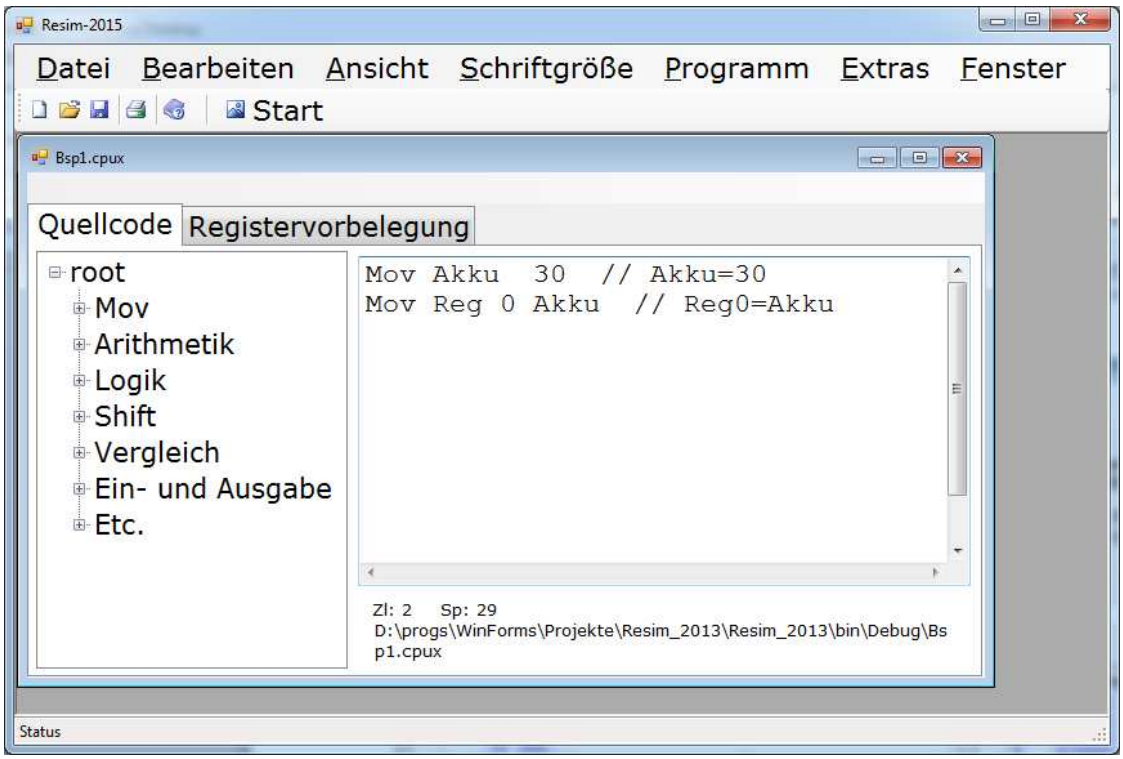


Abbildung 2 Oberfläche von Resim

Die Abbildung 2 zeigt das Hauptfenster von Resim. Das Programm -arbeitet mit Menüs, einer Schalterleiste und grafischen Elementen zur Anzeige der Inhalte. Die einzelnen Befehle können mit Hilfe der Baumstruktur ausgewählt werden.

2.2 Resim-Befehle

Resim-Befehle

Transportbefehle

- MOV Akku {Konstante, Register}
  - Mov Akku 3 — 3 in den Akku
  - Mov Akku Reg 5 — Inhalt von Reg 5 in den Akku
- MOV {Register} Akkumulator
  - Mov Reg 5 Akku — Akku in das Reg 5

Arithmetikbefehle

- ADD Akku {Konstante, Register}
  - ADD Akku 44 — Addiert 44 zum Akku
  - ADD Akku Reg 3 — Addiert den Inhalt von Reg 3 zum Akku
- SUB Akku {Konstante, Register}
  - SUB Akku 44 — Subtrahiert 44 vom Akku
  - SUB Akku Reg 3 — Subtrahiert Reg 3 vom Akku
- MOD Akku {Konstante, Register}
  - MOD Akku 44 — Berechnet den Rest von Akku und 44
  - MOD Akku Reg 3 — Berechnet den Rest von Akku und Reg 3
- MULT Akku {Konstante, Register}
  - MULT Akku 44 — Multipliziert 44 mit dem Akku
  - MULT Akku Reg 3 — Multipliziert Reg 3 mit dem Akku
- DIV Akku {Konstante, Register}
  - DIV Akku 44 — Dividiert den Akku durch 44
  - DIV Akku Reg 3 — Dividiert den Akku durch Reg 3
- INC Akku — Erhöht den Akku um eins
- DEC Akku — Verringert den Akku um eins

Logik-Befehle

- AND Akku {Konstante, Register}
  - AND Akku 44 — Bitweises logisches Und Akku mit 44
  - AND Akku Reg 3 — Bitweises logisches Und Akku mit Reg 3
- OR Akku {Konstante, Register}
  - OR Akku 44 — Bitweises logisches Oder Akku mit 44
  - OR Akku Reg 3 — Bitweises logisches Oder Akku mit Reg 3
- XOR Akku {Konstante, Register}
  - XOR Akku 44 — Bitweises logisches XOR Akku mit 44
  - XOR Akku Reg 3 — Bitweises logisches XOR Akku mit Reg 3
- NOT Akku — Bitweises negieren des Akkus

Schiebe-Befehle

- SHL Akku {Konstante, Register}
  - SHL Akku 2 Verschieben des Akkus um 2 Bit nach links
  - SHL Akku Reg 3 Verschieben des Akkus um Reg 3 Bit nach links
- SHR Akku {Konstante, Register}
  - SHR Akku 2 Verschieben des Akkus um 2 Bit nach rechts
  - SHR Akku Reg 3 Verschieben des Akkus um Reg 3 Bit nach rechts
- ROL Akku Konstante
  - ROL Akku 1
    - Verschieben des Akkus um 1 Bit nach links, das MSB wandert in das carry-Flag. Das LSB erhält den vorherigen Wert des Carry-Flags
- ROR Akku Konstante
  - ROR Akku 1
    - Verschieben des Akkus um 1 Bit nach rechts, das MSB erhält den Wert des Carry-Flags, das LSF wandert in das carry-Flag.
  -

Beispiele:

- `00011001 SHL ergibt 00110010`
- `00011001 SHR ergibt 00001100`
- `00011001 ROL ergibt 00110010 wenn Carry nicht gesetzt`
- `00011001 ROL ergibt 00110011 wenn Carry gesetzt`
- `00011001 ROR ergibt 00001100 wenn Carry nicht gesetzt`
- `00011001 ROR ergibt 10001100 wenn Carry gesetzt`

Sprungbefehle

- CMP AKKU {Konstante, Register}
  - CMP AKKU 4
    - Vergleicht den Akku mit einer Konstanten. Die beiden Werte werden subtrahiert.
  - CMP Reg 2
    - Vergleicht den Akku mit dem Inhalt des Registers 2. Die beiden Werte werden subtrahiert.
- 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 in E/A Einheit
  - LiesZahl "Eingabe"
- IN {Akku, Register}
  - IN Akku — Kopiert den Wert aus der E/A-Einheit in den Akku
  - IN Reg 3 — Kopiert den Wert aus der E/A-Einheit in Reg 3
- OUT {Akku, Register}
  - OUT Akku — Ausgabe des Akkus in den Editor
  - OUT Reg 3 — Ausgabe des Reg 3 in den Editor
- OUT NL — Neue Zeile
- OUT String "Ergebnis" — Ausgabe eines Textes

Eingabedialoge im Debugger:

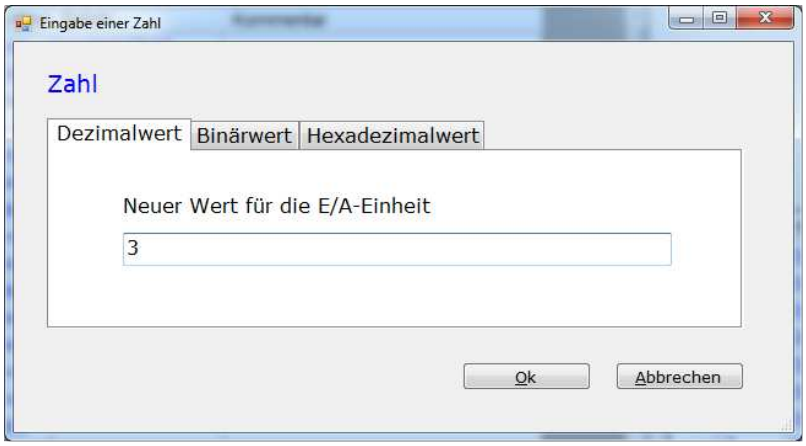


Abbildung 3 Eingabe einer dezimalen Zahl im Debugger

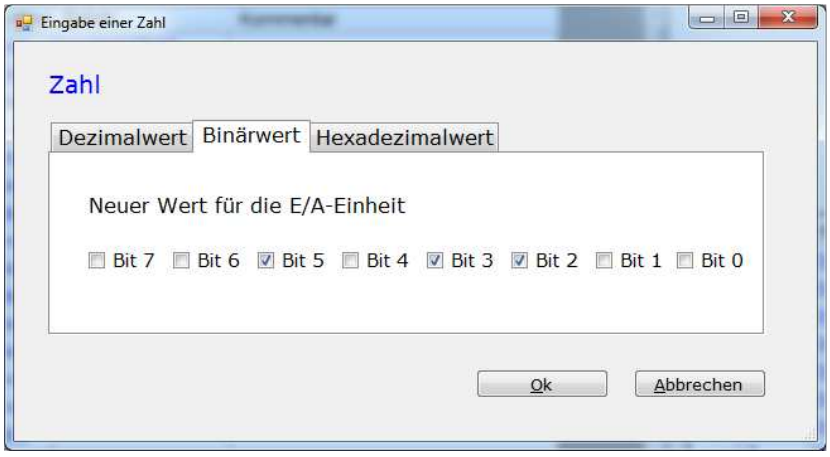


Abbildung 4 Eingabe einer binären Zahl

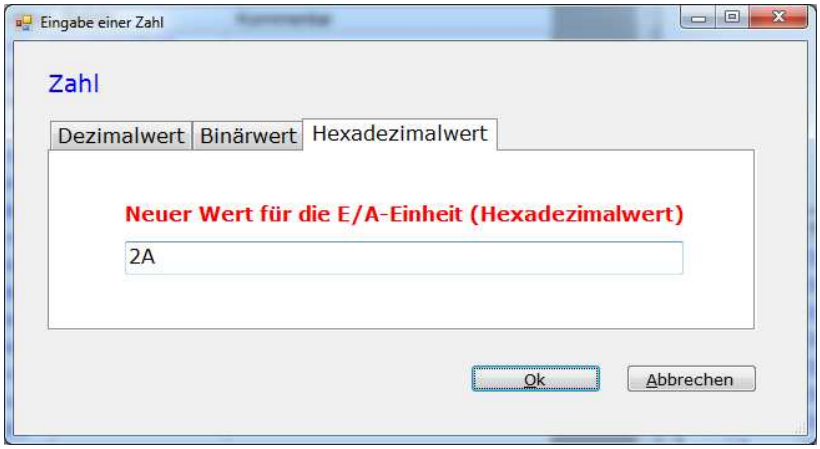


Abbildung 5 Eingabe einer hexadezimalen Zahl

Steuerbefehle

- Clear Carry
  - Löscht das Carry-Flag
- Goto Sub / IRET
- NOP (No Operation)

1 Aufgaben

1.1 Addieren zwei eingegebene Zahlen

Java:

```
int a;
int b;
int c;

a= readZahl();
b= readZahl();

c=a+b;

Syso("Ergebnis ist"+c);
```

Resim:

```
Register 0          a
Register 1          b
Register 2          c

    LiesZahl "Eingabe a"
    In Reg 0
    LiesZahl "Eingabe b"
    In Reg 1
    Mov Akku Reg 0              // akku=a
    Add akku reg 1              // akku=akku+b
    Mov reg 2 akku              // c=akku

    Out NL                  // NewLine
    Out String "Ergebnis"
    Out reg 2
```

1.2 Berechne die Summe von 1 bis n

$$s = \sum_{i=1}^{n} i$$

Java:

```
int s=0;
int i=1;
int n=10;

while (i<=n) {
    s+=I;
    i++;
}
System.out.Println("Ergebnis ist: "+s);
```

Resim:

```
Register 0      n
Register 1      i       Laufindex von 1 bis n
Register 2      summe

        LiesZahl "Eingabe n"
        In Reg 0
        Mov Akku 0              // s=0
        Mov Reg 2 Akku
        Mov Akku 1              // i=1
        Mov Reg 1 Akku
M1:     mov akku reg 2          // akku=s
        Add akku reg 1          // akku=akku+i
        Mov reg 2 akku          // s=akku

        Mov akku reg 1          // akku=i
        Inc akku                // akku++
        Mov reg 1 akku          // i=akku

        Mov akku reg 0          // akku = n
        Cmp akku reg 1          // akku = n – i
        Jge M1                  // Sprung, wenn n>=i

        Out NL                  // NewLine
        Out String "Ergebnis"
        Out reg 2
```

2 CPU-Fenster

Mit der Taste „F5“ wechselt man in den CPU-Dialog.

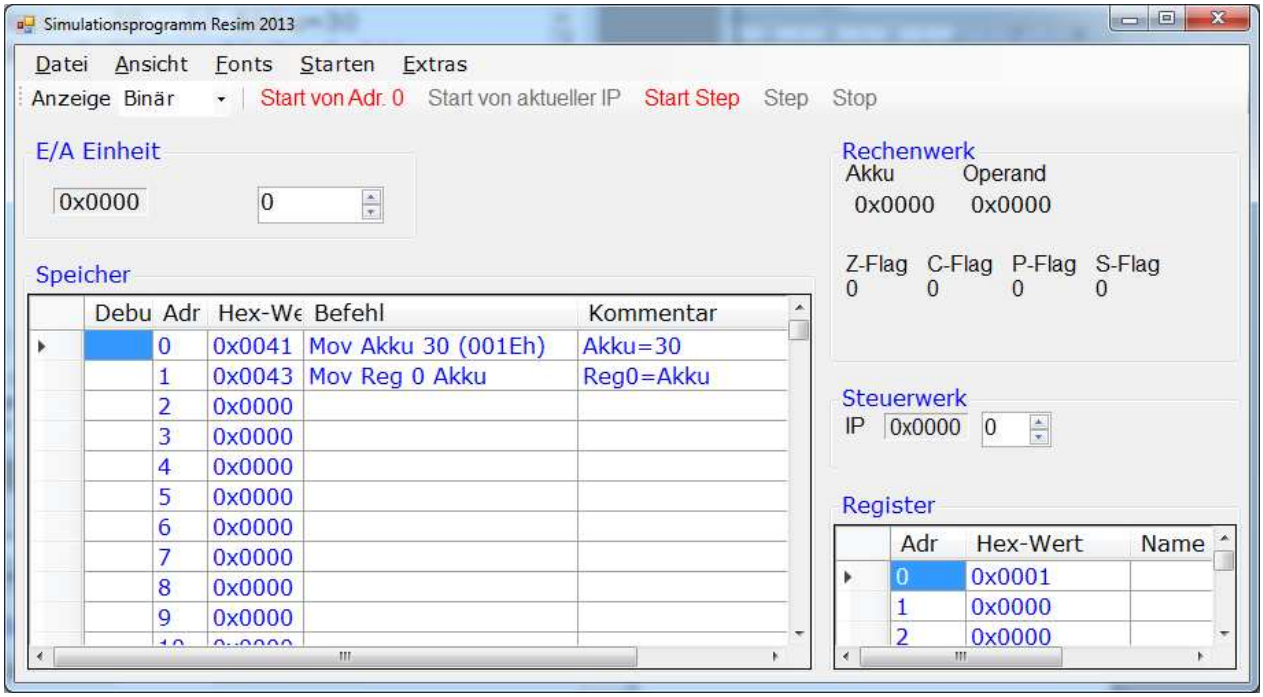


Abbildung 6 Anzeige des Debugger, CPU, Programm Bsp1.cpux

2.1.1 Menüstruktur

2.1.1.1 Menü Datei

Eintrag	**Beschreibung**
Schließen	Beendet das Programm.

2.1.1.2 Ansicht

Eintrag	**Beschreibung**
Ausgabefenster	Zeigt das Ausgabefenster an. Eigentlich ein einfacher Texteditor, in dem mittel dem Befehl „Out“ Werte ausgegeben werden können.
Löschen aller Halte-punkte	Mit diesem Menüeintrag werden alle Haltpunkte, Breakpoints, gelöscht.

2.1.1.3 Fonts

Mit diesen Menüeinträgen kann die Schriftgröße der grafischen Elemente setzen.

2.1.1.4 Starten

Eintrag	Beschreibung
Starten von IP=0	Startet das Programm vom Anfang. Der Shortkey ist F5.
Debug von IP=0	Startet das Programm im Debug-Modus vom Anfang. Im Debugmodus kann man jeden Schritt einzeln abarbeiten. Der Shortkey ist Strg+F5.
Start von akt. IP	Startet das Programm von der aktuellen Kursorposition.
Startet Einzelschritt	Startet den Einzelschrittmodus. Mit dem nächsten Befehl kann man jeden Schritt einzeln abarbeiten. Wird auch im Debugger-Modus benötigt.
Einzelschritt	Mit diesem Befehl kann man jeden Schritt einzeln abarbeiten.
Stop	Hält das Programm an.

2.1.1.5 Extras

Eintrag	Beschreibung
Taschenrechner	Dieser startet den Taschenrechner.
Extras	Mit dem Menüeintrag kann das Timerintervall gesetzt werden.

2.2 Grafische Elemente

Folgende Grafische Elemente sind im Dialogfenster vorhanden:

2.2.1.1 Ein- und Ausgabeeinheit

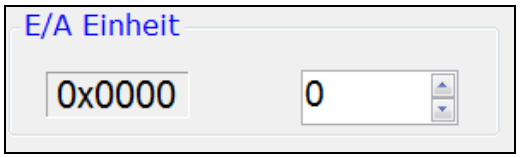


Abbildung 7 E/A-Einheit

Zeigt die Ein- und Ausgabeeinheit. Mit dem rechten Editor kann eine Zahl eingestellt werden. Die Zahl wird links in der hexadezimalen Darstellung angezeigt. Wenn man mit der Maus über die Zahl geht, erscheint eine dezimale Anzeige.

2.2.1.2 Speicher

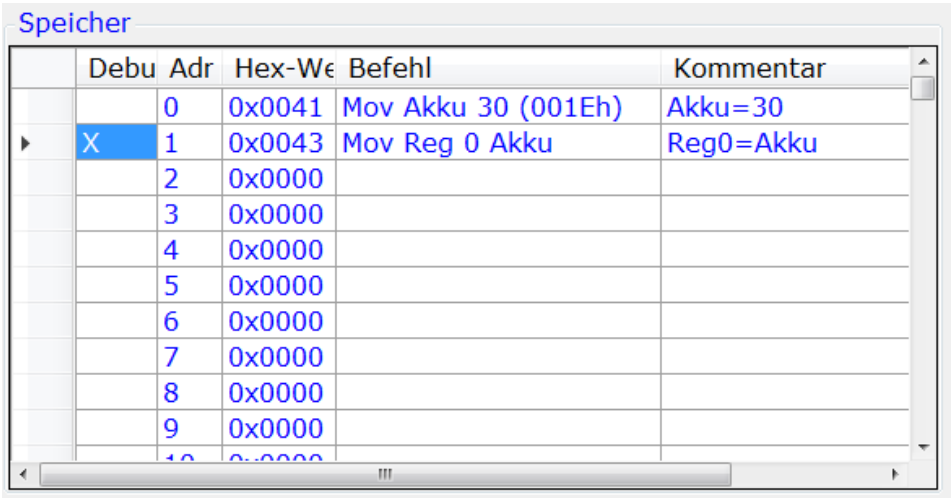


Abbildung 8 Anzeige des Speichers mit den Befehlen

Im Speicher sind alle Befehle mit den Kommentaren abgebildet. Mit einem Doppelklick der Maus kann man einen Haltepunkt setzen. Dieser wirkt sich aber nur im Debug-Modus aus.

2.2.1.3 Register

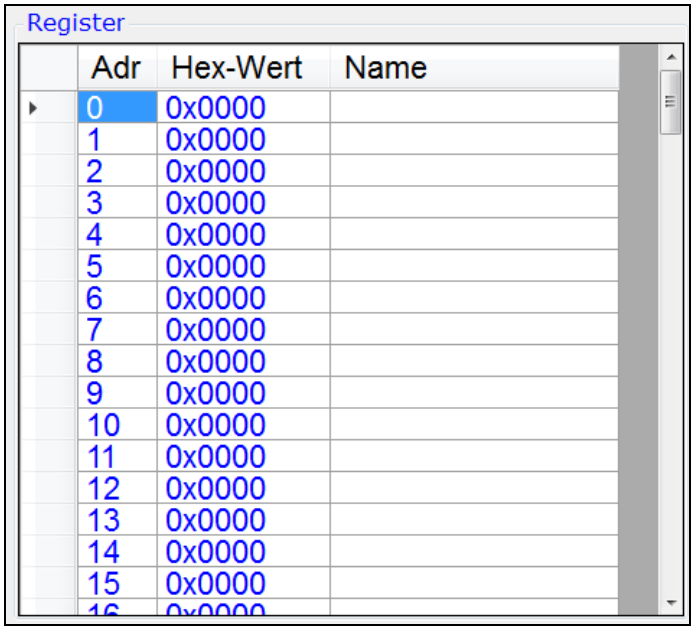


Abbildung 9 Register

Zeigt die Register von 0 bis 19. Mit einem Doppelklick kann man den Inhalt bzw. die Bezeichnung ändern. Man schreibt einfach die neuen Werte in die Zelle.

2.2.1.4 Steuerwerk

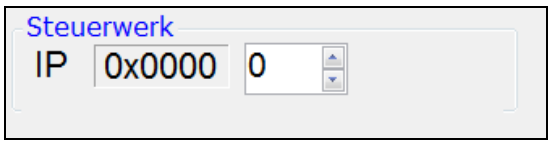


Abbildung 10 Steuerwerk

Das Steuerwerk zeigt den aktuellen Befehlszeiger, Instruktionspointer.

2.2.1.5 Rechenwerk

Zeigt das Rechenwerk. Im Akku steht immer der erste Operand. Hat man zwei Operanden, so wird die Anzeige „Operand“ benutzt. Die ALU ist hier nur als Panel angedeutet.

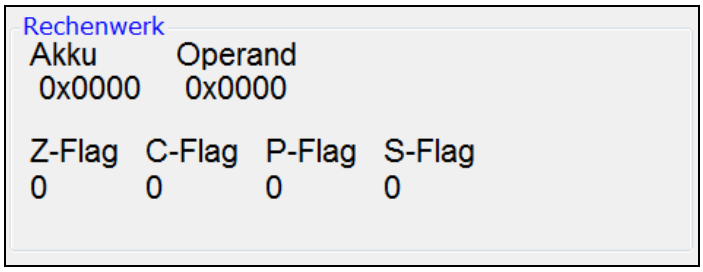


Abbildung 11 Rechenwerk

Das Ergebnis des Befehls wird wieder in den Akkumulator eingetragen. Dazu werden die Flags Zero, Carry, Parity und Sign gesetzt. Das Zero-Flag wird gesetzt, wenn das Ergebnis Null ist. Das Parity-Flag ist ein Zusatzbit, um Datenübertragungsfehler zu erkennen. Das Carry-Flag, Übertragungsbit, wird gesetzt, wenn das Ergebnis größer als 65535 ist. Es dient dazu, Ketten von Operationen korrekt abzubilden. Das Carry-Bit wird dann in der nächsten Operation benutzt. Ein Beispiel ist die Multiplikation mittels ROR, SHL und dem Carry-Flag.

3 Befehlsgruppen

Dieses Kapitel zeigt, wie man die einzelnen Befehle in sinnvolle Gruppen resp. Aufgaben, einteilen kann. Es gibt immer wieder „Teilaufgaben“, die in den Programmen gelöst werden müssen. Dieses Kapitel zeigt für die wichtigsten Teilaufgaben die Lösungen.

3.1 Befehlsgruppen

Dieses Kapitel zeigt, wie man die einzelnen Befehle in sinnvolle Gruppen resp. Aufgaben, einteilen kann. Es gibt immer wieder „Teilaufgaben“, die in den Programmen gelöst werden müssen. Dieses Kapitel zeigt für die wichtigsten Teilaufgaben die Lösungen.

3.2 Einlesen einer Zahl

- LiesZahl "x"
- In Reg 5

Eine Zahl wird in die Eingabeeinheit eingelesen. Danach wird es in das Register fünf eingetragen.

3.3 Register mit einem Wert belegen

Diese Aufgabe kann man nur über den Akkumulator realisieren:

- Mov Akku Konstante Mov Akku 42
- Mov Reg(i) Akku Mov Reg 5 Akku

Das Register und die Konstante muss angegeben werden.

3.4 Operation mit einem Register und die Speicherung in ein neues Register

Diese Aufgabe kann man nur über den Akkumulator realisieren:

- Mov Akku Reg(i) // Register in den Akkumulator
- Operation Akku Reg(j) // add, sub, div, mod, mult
- Mov Reg(k) Akku // Wert in das Zielregister schreiben

Beispiel:

- Mov Akku Reg 3 // Akku erhält den Wert des Registers 3
- Add Akku Reg 2 // Zum Akku wird der Inhalt des Reg 2 addiert
- Mov Reg 4 Akku // Ergebnis wird im Register 4 gespeichert

- Mov Akku Reg(i) // Register in den Akkumulator
- Operation Akku 55 // add, sub, div, mod, mult
- Mov Reg(k) Akku // Wert in das Zielregister schreiben

Beispiel:

- Mov Akku Reg 3 // Akku erhält den Wert des Registers 3
- Add Akku 42 // Zum Akku wird die Zahl 42 addiert
- Mov Reg 4 Akku // Ergebnis wird im Register 4 gespeichert

Die Register i,j,k müssen ausgewählt werden. Können aber auch teilweise identisch sein

3.5 Sprung, wenn der Akku größer als der Inhalt des Registers 2 ist

- Mov Akku 42
- CMP Akku Reg 2 // in Akku steht die Differenz „Akku-Reg2“
- JG Akku Marker1 // Jump greater, wenn Differenz > 0

Die Sprungmarke wird mit einem Namen ganz links mit einem Doppelpunkt eingetragen.
Beispiel:
Marker1: NOP

3.6 Sprung, wenn der Akku größer gleich Null ist

```
M1:
    // Anweisungen
    Mov  Akku  42          // Beispiel oder Reg 1
    CMP  Akku  Reg 3       // in Akku steht die Differenz  „42-Reg 3“
    JGE  M1                // Jump greater equal          if (42>=Reg3)
```

3.7 Sprung, wenn der Akku kleiner Null ist

```
M1:
    // Anweisungen
    Mov  Akku  Reg 3       // Akku=Reg 3
    CMP  Akku  42          // in Akku steht die Differenz  „Reg3-42“
    JL  M1                 // Jump less than          if (Reg3<42)
```

3.8 Sprung, wenn der Akku kleiner gleich Null ist

```
M1:
    // Anweisungen
    Mov  Akku  Reg 3       // Akku=Reg 3
    CMP  Akku  42          // in Akku steht die Differenz  „Reg3-42“
    JLE  M1                // Jump less equal          if (Reg3<=42)
```

3.9 Sprung, wenn der Akku ungleich Null ist

```
M1:
    // Anweisungen
    Mov  Akku  Reg 3       // Akku=Reg 3
    CMP  Akku  42          // in Akku steht die Differenz  „Reg3-42“
    JNZ  M1                // Jump Not Zero          if (Reg3<>42)
```

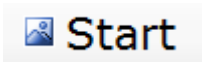
4 Beispiele

4.1 Kopieren einer Konstante in ein Register (Bsp1)

a) Neue Datei anklicken  und speichern unter Bsp1

b) Eintragen des Befehls:
- Mov Akku 122
- Mov Reg 0 Akku

Ein Register kann nur über den Akku einen Wert erhalten.
Mathematische Operationen laufen auch über den Akkumulator.

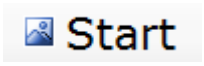
c) Starten des Programms mit der Taste F5 oder mit dem Schalter 

d) Der Akkumulator enthält den Wert 122_{10} bzw. 0x7A.

4.2 Kopieren und addieren von Konstanten (Bsp2)

a) Neue Datei anklicken  und speichern unter Bsp2

b) Eintragen folgender Befehle:
- Mov Akku 122
- Add Akku 35
- Mov Akku Reg 0

c) Starten des Programms mit der Taste F5 oder mit dem Schalter 

d) Der Akkumulator enthält den Wert 157_{10} bzw. 0x9D.

4.3 Einlesen einer Zahl, kopieren ins Register Null (Bsp3)

a) Neue Datei anklicken  und speichern unter Bsp3

b) Eintragen folgender Befehle:
- Lies Zahl "Zahl:"
- In Reg 0

4.4 Berechnen einer Formel (Bsp4)

Dieses Beispiel zeigt die Berechnung einer Multiplikation und einer Addition. Folgende Formel soll berechnet werden:

$c = a + 4 \cdot b$

a) Im ersten Schritt werden nun die Register verteilt
   a Register 0
   b Register 1
   c Register 2

b) Nun wird der Algorithmus entwickelt.
   In einer Hochsprache kann man die obige Gleichung direkt eingeben. In Assembler bzw. ReSim ist das nicht möglich. Da ReSim keine Punkt- vor Strichrechnung berücksichtigt, muss man erst den Ausdruck „4 mal b“ berechnen und dann a dazu addieren.

c) Neue Datei anklicken  und speichern unter Bsp4

Einlesen von a:

- Lies Zahl "a:"
- In Reg 0

Einlesen von b:

- Lies Zahl "b:"
- In Reg 1

Berechnen:

- Mov Akku Reg 1 // Zahl b
- SHL Akku 2 // Damit wird die Zahl b um 2 Bits nach links verschoben,
  also $2^2 = 4$
- Add Akku Reg 0 // a !

Ergebnis speichern:

- Mov Reg 2 Akku
- OUT Akku // Damit wird das Ergebnis ausgegeben

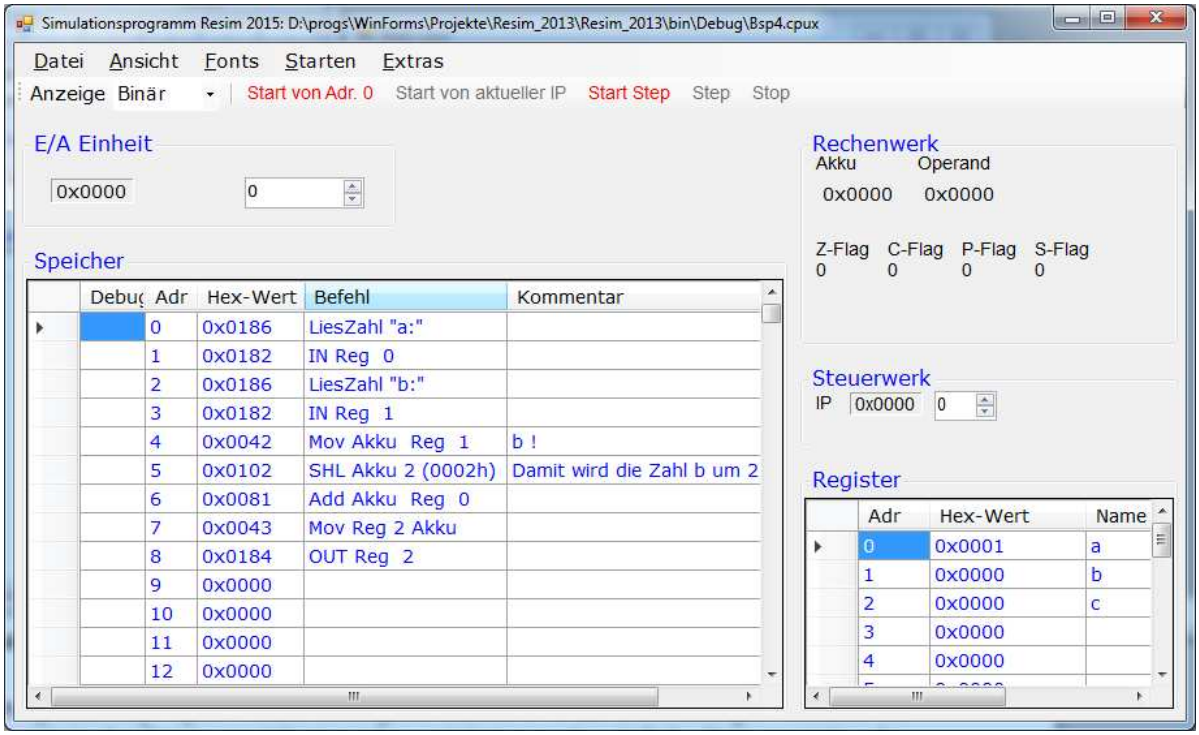


Abbildung 12 Programm Bsp4.cpux

Im Programm Bsp4.cpux wurden die Register null bis zwei mit Bezeichnungen versehen. Da diese mit in der Datei gespeichert werden, existiert nun eine einfache Dokumentation des Algorithmus.

4.4.1 Eingaben:

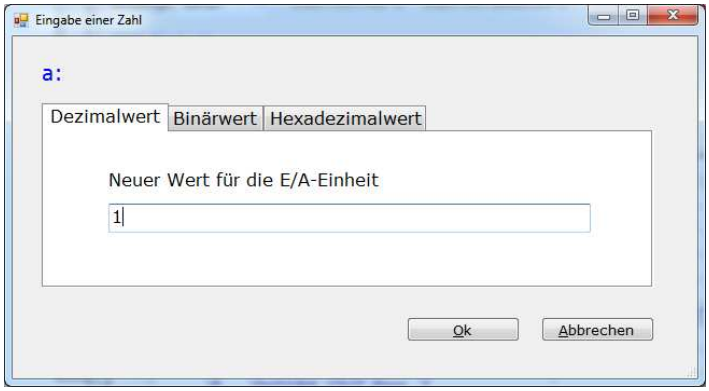


Abbildung 13 Eingabe der 1. Zahl a

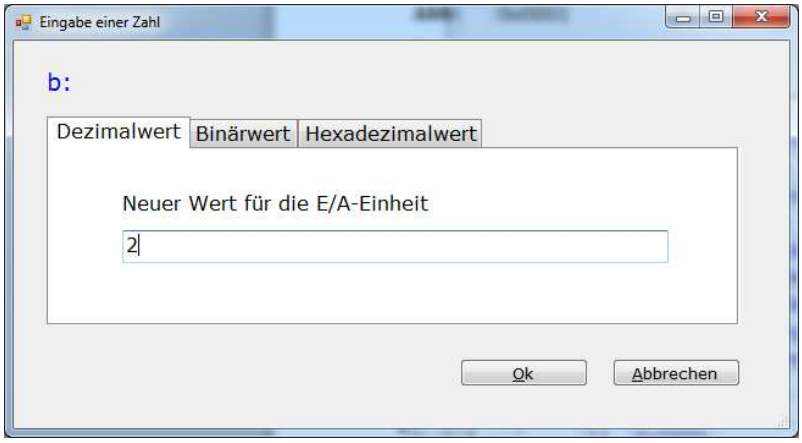


Abbildung 14 Eingabe der 2. Zahl b

Anzeige der Ein- und Ausgaben im Dialog:

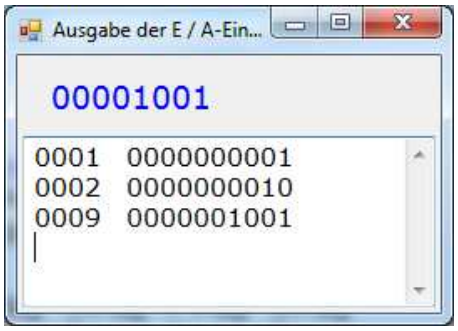


Abbildung 15 Ausgabedialog

Weitere Aufgabe:

Erstellen eines ReSim-Programms mit Einlesen und Speichern von sechs Zahlen und Berechnen folgender Formel

$$g = \frac{a+b}{c+d} - \frac{a-b}{e+f}$$

4.5 Bestimmen des Maximum zweier Zahlen

```
LiesZahl  "a: "
in  Reg  0
LiesZahl  "b:  "
in  Reg  1
mov  Akku  Reg  0
cmp  Akku  Reg  1  // akku=reg0-Reg1
jg  a_greater_b
  // b greater a
 mov  akku  reg  1
 mov  reg  2  akku
```

```
  jmp weiter

a_greater_b:
  mov akku reg 0
  mov reg 2 akku

weiter:
  out reg 2
```

4.6 Setzen von Bits (Bsp5)

Dieses Beispiel zeigt die Bit-Manipulationsmöglichkeiten. Folgende Aufgabe besteht:

- Einlesen einer Zahl in Register 0
- Setzen des vierten Bits[3]
- Speichern in Register 1

a) Erst wird der Algorithmus entwickelt.
Um das vierte Bit zu setzen, muss man die Oder-Verknüpfung aufrufen. Der Wert ermittelt sich aus der Nummer, indem man die Zweier-Potenz ausrechnet. Zwei hoch vier ist 16.

b) Neue Datei anklicken  und speichern unter Bsp4

Einlesen von X:

- „Lies Zahl in E/A-Einheit“
- „In Akku“
- „Mov Register, Akku“, Registernummer 0

Berechnen:

- „Mov Akku, Register“, Registernummer 0
- „ODER Akku mit Konstante 16“

Ergebnis speichern:

- „Mov Register, Akku“, Registernummer 1
- „OUT Akku“ // Damit wird das Ergebnis ausgegeben

Programm:

Adr	**Binärwert**	**Befehl / Daten**
000	10111100	Lies Zahl in E/A-Einheit
001	11000000	IN Akku
002	00101000	Mov Akku, Register
003	01111111	OR Akku, Konstante
004	00010000	16
005	00110001	Mov Reg, Akku
006	11001000	OUT Akku

[3] Informatiker zählen von Null, das 4. Bit ist „normal“ das fünfte Bit

4.6.1 Abfragen von Bits (Bsp6)

Dieses Beispiel zeigt die Bit-Manipulationsmöglichkeiten.
Folgende Aufgabe besteht:

- Einlesen einer Zahl in Register 0
- Testen, ob das dritte Bit gesetzt ist
- Falls ja, wird eine eins ins Register 1 gespeichert
- Falls nein, wird eine 1 ins Register 0 gespeichert
- Ausgabe des Register 1

a) Erst wird der Algorithmus entwickelt.
Um das dritte Bit zu selektieren, muss man die Und-Verknüpfung aufrufen. Alle Bits, die nicht interessant sind, erhalten eine Null, alle anderen eine Eins. Für die obige Aufgabe ermittelt man den Wert Acht.

b) Neue Datei anklicken  und speichern unter Bsp5

Einlesen von X:

- LiesZahl "a: "
- In Reg 0

Berechnen:

- mov Akku Reg 0
- and Akku 8

Sprung:

Nun muss der Akkumulatorwert abgefragt werden. Ist er größer als Null, so ist das Bit gesetzt, und man muss eine Eins ins Register 1 eintragen. Sonst eine Null. Da es keine Einrückungen à la Java oder C gibt, muss man zweimal „springen“

Algorithmus:

```
        AND  Akku  8          // nun wird dieser Wert abgefragt, kein CMP !
        jg  L1                // wenn Akku > 0 Sprung nach L1
        mov  Akku  0
        jump  L2              // unbedingter Sprung

L1:     mov  Akku  1

L2:     mov  Reg 1  Akku      // kann für beide Fälle programmiert werden
        out Akku
```

Hinweise:

- Die L1 und L2 sind Sprungmarken (Label).

Programm:

```
LiesZahl "a: "
 In Reg 0

 mov akku Reg 0
 and akku 8  // 3. Bit, 2^3 = 8
 jg  L1                    // wenn Akku > 0 Sprung nach L1
 mov  Akku  0
 jump  L2     // unbedingter Sprung

L1:    mov  Akku  1

L2:    mov  Reg 1  Akku              // kann für beide Fälle programmiert werden
    out Akku
```

123 liefert eine eins
37 liefert eine null

Weitere Aufgaben:

- Eingabe einer Zahl, Löschen von Bit fünf
- Eingabe einer Zahl, setzen von Bit drei und fünf
- Eingabe einer Zahl, testen der Bits zwei und sechs

4.7 Berechnen einer Summe (Bsp8)

Dieses Beispiel zeigt die Schleifen-Technik. Folgende Aufgabe besteht:

- Einlesen einer Zahl n in Register 0 // n
- Aufsummieren der Zahlen von 1 bis n
- Ausgabe der Summe

a) Algorithmus erstellen
Bei dieser Aufgabe muss man als erstes die Register zuordnen. Danach bedingt die verwendete Schleife die Initialisierung der Zählvariablen.

Grobe Skizze des Algorithmus:

- Einlesen einer Zahl
- Kopieren ins Register 0
- Addieren um eins
- Kopieren ins Register 1 // Abfrageregister jump Akku < Reg(1)
- Löschen des Registers 2 // Summenregister
- Setzen des Register 3 auf eins // Zähler
  // Schleife L1:
- kopiere Register 3 in den Akku // summe = summe + Zähler
- Addiere Register 2 zum Akku
- Speichere Akku in Register 2

- Kopiere Zähler in den Akku

- Addiere um Eins
- Kopiere Akku in den Zähler

- Abfrage und Sprung: Wenn Zähler

b) Neue Datei anklicken  und speichern unter Bsp5

Einlesen von N:

- „Lies Zahl in E/A-Einheit“
- „In Akku“
- „Mov Register, Akku“, Registernummer 0
- „Addiere Akku“ um eins
- „Mov Register, Akku“, Registernummer 1

Initialisierung der Summe und des Zählers:

- „Mov Akku, Konstante“, Null
- „Mov Register, Akku“, Registernummer 2
- „Mov Akku, Konstante“, Eins
- „Mov Register, Akku“, Registernummer 3

Schleife: (Adresse 12)

- L1: „Mov Akku, Register“, Registernummer 2“ // Zähler
- „OUT Akku“ // Kontrolle
- „Add Akku, Register“, Registernummer 3“ // summe
- „Mov Register, Akku“, Registernummer 2 // summe=summe+Zähler
- „Mov Akku, Register“, Registernummer 3“
- „Add Akku, Konstante“, Eins
- „Mov Register, Akku“, Registernummer 3 // Zähler = Zähler+1

Sprung:

Nun muss der Akkumulatorwert mit dem Register 1 (n+1) überprüft werden.

- „Jump zur Adresse, wenn Akku < Register“, Registernummer 2“ // summe
  - Jump Akku LT Reg(i)
  - Adresse 12
  - Register 1

Programm:

Adr	**Binärwert**	**Befehl / Daten**
000	10111100	Lies Zahl in E/A-Einheit
001	11000000	IN Akku
002	00110000	Mov Reg, Akku
003	01011000	Add Akku, Konstante
004	00000001	1
005	00110001	Mov Reg, Akku
006	00111111	Mov Akku, Konstante
007	00000000	0
008	00110010	Mov Reg, Akku

```
009  00111111   Mov Akku, Konstante
010  00000001   1
011  00110011   Mov Reg, Akku
012  00101011   Mov Akku, Register
013  11001000   OUT Akku
014  01001010   Add Akku, Register
015  00110010   Mov Reg, Akku
016  00101011   Mov Akku, Register
017  01011000   Add Akku, Konstante
018  00000001   1
019  00110011   Mov Reg, Akku
020  11110001   Jump Akku LT Reg(i) zu Adresse
021  00001100   12
022  00101010   Mov Akku, Register
023  11001000   OUT Akku
```

Weitere Aufgaben:

Summen-Bestimmung:
- Einlesen zweier Zahlen
- Ausgabe der Summen beider Zahlen (out)

Mittelwert-Bestimmung:
- Einlesen zweier Zahlen
- Ausgabe des Mittelwertes, ganzzahlige Division (out)
- Einlesen einer Zahl n
- Berechnen der Summe von 1 bis n
- Ermitteln des Mittelwertes
- Ausgabe des Mittelwertes

Teiler einer Zahl
- Einlesen einer Zahl x
- Berechnen und Ausgabe aller echten Teiler der Zahl x

Test auf vollkommene Zahl
- Einlesen einer Zahl x
- Berechnen der Summe aller echten Teiler der Zahl x
- Ist die Summe gleich der Zahl x
- dann Ausgabe einer 1
- sonst Ausgabe einer 0
- Lösung: 33 Zeilen

4.8 Fibonacci-Folge:

Einlesen einer Zahl N
Ermitteln der Fibonacci-Folge bis die n-te Zahl ausgegeben wurde
Lösung: 27 Zeilen
Maximal kann n den Wert 13 haben

Mögliche Registerbelegeung:
Register 0: N // wie viele Folgenglieder werden ausgegeben
Register 1: a
Register 2: b
Register 3: c=a+b
Register 4: Zähler

Die Fibonacci-Folge fängt bei 0 und 1 an.
Die nächste Zahl ergibt sich aus der Summe der vorherigen **beiden** Zahlen

Lösung:

```
// berechnet die Fibonacci-Folge
// 0 1 1 2 3 5 8 13 21 34 55
LiesZahl  "Maximal Wert"
in reg 0

// init
out nl

mov akku 0
mov reg 1 akku    // a=0
out akku

mov akku 1
mov reg 2 akku    // b=1
out akku

loop: mov akku reg 1   // akku=a
   add akku reg 2    // akku=a+b
   mov reg 3 akku    // Zwischenspeicher c=a+b
   out reg 3

   mov akku reg 2
   mov reg 1 akku  // a=b
   mov akku reg 3
   mov reg 2 akku // b=c

   mov akku reg 0  // n
   cmp akku reg 2  // akku=n-b
   jg loop
```

Anzeige im Editor

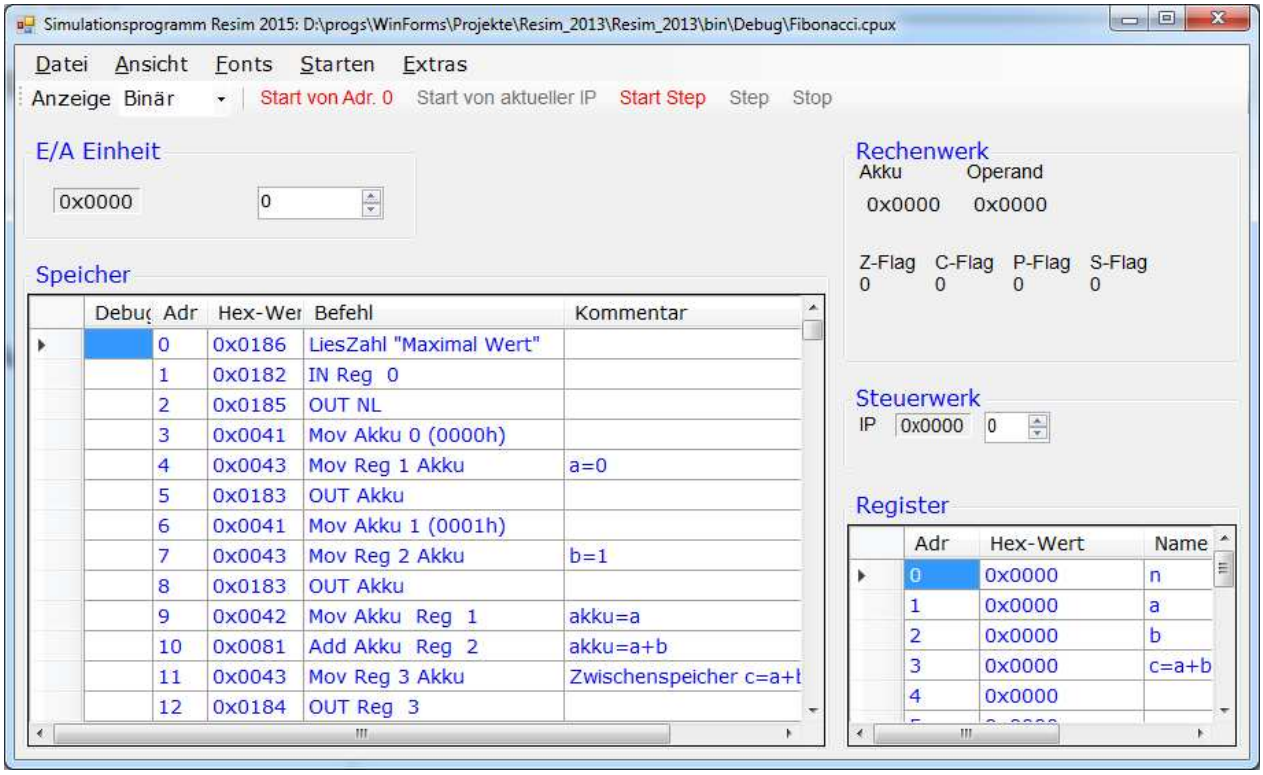


Abbildung 16 Fibonacci

Ergebnis im Ausgabefenster:

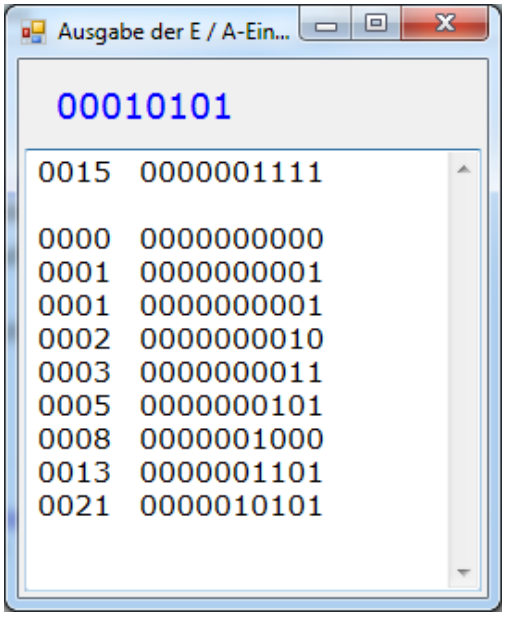


Abbildung 17 Anzeige der ersten 15 Fibonacci-Zahlen

Weitere schwere Aufgaben:

Primzahl-Test:
Einlesen einer Zahl x
Wenn x eine Primzahl ist
dann Ausgabe von 1
sonst Ausgabe von 0

Primzahl-Ausgabe:
Einlesen einer Zahl x
Bestimmen und Ausgabe aller Primzahlen von zwei bis zur Zahl x

Fakultät:
Einlesen einer Zahl x
Ausgabe der Fakultät
Maximal darf aber fünf eingeben werden
33 Zeilen mit der einfachen Lösung

Teiler einer Zahl:
Einlesen einer Zahl x
Berechnen und Ausgabe aller echten Teiler der Zahl x
Hier muss die Multiplikation die Modulo-Funktion ersetzen, da die Modulo-Funktion nur mit einer Konstanten implementiert wurde.

5 Grundlagen

5.1 Logische Verknüpfungen

Oder Verknüpfung:

ODER	**0**	**1**
0	0	1
1	1	1

Und Verknüpfung:

UND	**0**	**1**
0	0	0
1	0	1

Negation:

	0	**1**
NOT	1	0

Exklusives Oder:

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

XOR ergibt eine 1, wenn die Bits unterschiedlich sind!

5.1.1 Beispiele

Diese logischen Verknüpfungen werden auch auf Bitfolgen angewandt.

0001 ODER 1000 = 1001
0001 OR 1000 = 1001
0001 ∨ 1000 = 1001
0001 | 1000 = 1001

1011 UND 1000 = 1000
1011 AND 1000 = 1000
1011 ∧ 1000 = 1000
1011 & 1000 = 1000

NOT 1010 = 0101

NICHT 1010 = 0101
¬ 1010 = 0101
! 1010 = 0101

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

5.2 Zahlendarstellungen

Neben dem Dezimalsystem sind alle weiteren Zahlensysteme, die vor allem in der Informatik verwendet werden, sogenannte Stellenwertsysteme.
Definition - Stellenwertsystem:
Jede reelle Zahl wird durch eine Folge von Ziffern beschrieben:
Wie eine Ziffer zum Wert der Zahl beiträgt, hängt von ihrer Position bezüglich des Kommas ab. Sie wird dazu mit (B = Basis des Zahlensystems multipliziert. Für den Wert der Zahl ergibt sich damit:

$$X = \pm(Z_m B^m + Z_{m-1} B^{m-1} + ... + Z_1 B^1 + Z_0 B^0 + Z_{-1} B^{-1} + Z_{-2} B^{-2} ...)$$

$$X = \pm \sum_{\mu=-\infty}^{+\infty} Z_\mu B^\mu \approx \pm \sum_{\mu=-m}^{+n} Z_\mu B^\mu$$

Beispiel:
123,4 = 1·102 + 2·101 + 3·100 + 4·10-1

Zahlensystem	**Zahlenbasis**	**Ziffern**	**Beispiel**
Dualsystem	B=2	0, 1	11111001100
Fünfersystem	B=5	0, 1, 2, 3, 4	30441
Siebenersystem	B=7	0, 1, 2, 3, 4, 5, 6	5551
Oktalsystem	B=8	0, 1, 2, 3, 4, 5, 6, 7	3714
Dezimalsystem	B=10	0, 1, 2, 3, 4, 5, 6, 7, 8, 9	1996
Hexadezimalsystem	B=16	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F	7CC

Divisionsrestverfahren:

13 zur Basis 10 = 1101 zur Basis 2
denn:
13:2=6 Rest 1
6:2= 3 Rest 0
3:2= 1 Rest 1
1:2= 0 Rest 1

Das Konvertieren (Umwandeln) von Dezimalzahlen in Zahlen eines anderen Zahlensystems bzw. umgekehrt, oder zwischen den anderen Zahlensystemen, basiert auf der Definition der Stellenwertsysteme. Daraus wurde ein Konvertierungsverfahren, das „Divisions-Restwert-Verfahren“, entwickelt, das allgemein einsetzbar ist (es beruht darauf, dass man nach Abspalten von Resten sukzessive versucht, den neuen Basiswert aus der verbliebenen Zahl „auszuklammern“ und dann nach Potenzen des neuen Basiswertes zusammenfasst).

Beispiel - Umwandlung Dezimalzahl in Dualzahl: 4510 = X2 ?

45 : 2 = 22 Rest: 1, also 45 = 22 · 2 + 1
22 : 2 = 11 Rest: 0, also 45 = (11 · 2 + 0) · 2 + 1
11 : 2 = 5 Rest: 1, also 45 =
= ((5 · 2 + 1) · 2 + 0) · 2 + 1
= 5 · 23 + 1 · 22 + 0 · 21 + 1 · 20
5 : 2 = 2 Rest: 1
2 : 2 = 1 Rest: 0
1 : 2 = 0 Rest: 1

Ergebnis: 4510 = 1011012

5.2.1 Zahlenkonvertierung zwischen 2, 8 und 16er-System

Berechnung des Binärsystems: $4711_{10} = 1001001100111_2$

Damit ist es trivial, aus der Binär-Lösung sofort die weiteren Zahlen zu berechnen. Man fasst drei oder vier Bits zusammen:

a) Dual nach Hexadezimal
- Berechnen der Dualzahl
- Zusammenfassen von 4 Bits, von rechts beginnend
- $0001001001100111_2 = 1\ 0010\ 0110\ 0111_2 = 0001\ 0010\ 0110\ 0111_2$
- Alle „vier Bits“ in eine hexadezimale Zahl umwandeln
- $1\ 2\ 6\ 7_{16}$

b) Dual nach Oktal
- Berechnen der Dualzahl
- Zusammenfassen von 3 Bits, von rechts beginnend
- $0001001001100111_2 = 1\ 001\ 001\ 100\ 111_2 = 001\ 001\ 001\ 100\ 111_2$
- Alle „drei Bits“ in eine hexadezimale Zahl umwandeln
- $1\ 1\ 1\ 1\ 4\ 7_8$

5.3 Einer-Komplement

Regeln zur Umwandlung:

- Die Subtraktion einer positiven Zahl wird auf die Addition des Komplements zurückgeführt.
- Das Komplement einer Zahl ist die bitweise Vertauschung von Nullen und Einsen.
- Falls ein Überlauf stattfindet, muss diese „Eins“ zum Ergebnis hinzuaddiert werden.

Beispiel:

```
   +43      01010112
-   27      00110112
---------------------------------
   +43      01010112
+  -27      11001002
---------------------------------
+     1  0001111
                 1
---------------------------------
Σ  16     00100002
```

5.4 B-Komplement bzw. 2er-Komplement

Regeln zur Umwandlung:

- Die Subtraktion einer positiven Zahl wird auf die Addition des 2-Komplements zurückgeführt.
- Das Komplement einer Zahl ist die bitweise Vertauschung von Nullen und Einsen. Danach die Addition von Eins.
- Ein Überlauf wird ignoriert.

Beispiel:

```
   +43      01010112
-  +27      00110112
---------------------------------
   +43      01010112
+  -27      11001012
---------------------------------
+     1  0010000
---------------------------------
Σ  16     00100002
```

6 Literatur

[1] Duden der Informatik Dudenverlag Mannheim

[2] Giloi, Wolfgang K. Rechnerarchitektur, Springer-Verlag, Berlin

[3] Oberschelp, W. Vossen, G. Rechneraufbau und Rechnerstruktur, Oldenbourg-Verlag

[4] http://de.wikipedia.org/wiki/Von-Neumann-Architektur

[5] Heinz-Peter Gumm und Manfred Sommer, Einführung in die Informatik (Gebundene Ausgabe - 20. Oktober 2008), Verlag: Oldenbourg; Auflage: 8., vollst. überarb. Auflage. (20. Oktober 2008), ISBN-10: 3486587242, ISBN-13: 978-3486587241

[6] Script der Vorlesung

7 Indexverzeichnis

O

R

S

Z