Aufbau und Funktionsweise eines von-Neumann-Rechners

Ein Kurs mit praktischen Übungen am Simulator „Johnny“

Version 3.0

von Peter Dauscher

`peter.dauscher@gmail.com`

Inhaltsverzeichnis

Rechtliche Hinweise

1 Vorwort

Computer haben sich seit ihren Kindertagen sehr verändert: die Rechner sind immer leistungsfähiger, immer kleiner und immer billiger geworden. Die Rechenleistung, die in den 1950er Jahren von Computer erbracht wurde, der mehrere Schränke füllte, steckt heute in einem Mobiltelefon.

Etwa seit den 1980er Jahren ist der Computer auf dem Siegeszug durch die Privathaushalte. Heute verwenden die allermeisten Leute auch in ihrem alltäglichen Leben Computer, die meisten allerdings, ohne auch nur ansatzweise zu wissen, wie dieses Gerät aufgebaut und wie es die Computerhardware fertigbringt, Software auszuführen und damit „zum Leben“ zu erwecken.

Natürlich sind Computer – gerade die heutigen – so kompliziert, dass eine umfassende Behandlung im Unterricht ganz unmöglich ist. Daher muss sich der Informatikunterricht in der Schule und auch dieses Skript auf die wesentlichen Bestandteile und Funktionsprinzipien einschränken, muss aber jede Menge – auch wichtiger Details – außer Acht lassen.

Wenn man zum Beispiel einen Personal Computer (PC) öffnet, so erkennt man den groben Aufbau. Wie der Computer wirklich funktioniert, kann man jedoch nicht erkennen: der Computer sieht im laufenden Betrieb praktisch genauso aus wie im ausgeschalteten Zustand – wenn man von der Kühlung absieht und vielleicht ein paar Kontrolllampen absieht. Um dennoch eine Vorstellung von den Abläufen zu haben, greifen wir hier auf ein typisches Mittel zurück: die Simulation. Ein Simulationsprogramm – „Johnny“ genannt – zeigt ein stark vereinfachtes Modell eines Rechners, an dem man jedoch viele wesentliche Funktionsabläufe erkennen kann.

Auf viele Details wie etwa der genaue Aufbau der Chipsätze oder modernere Entwicklungen, zum Beispiel die Mehrkern-Technologie kann und soll hier nicht eingegangen werden.

Viel Spaß und viel Erfolg beim Durcharbeiten dieses Skripts.

PS.:
Dieses Skript entstand im Jahr 2012. Sollten Sie dieses Skript zu einem sehr viel späteren Zeitpunkt lesen, werden Ihnen viele hier beschriebene Fakten vielleicht als antiquiert, oder sogar längst überholt vorkommen.

In diesem Fall: Schwelgen Sie ruhig in Nostalgie und freuen Sie sich an der museumsreifen Darstellung in diesem Skript. Und darüber, dass Informatik (oder wie immer diese Wissenschaft zum Zeitpunkt Ihres Lesens gerade heißen mag) keine statische Wissenschaft ist, sondern dass sie ein dynamischer Prozess ist, der von intelligenten Leuten vorangetrieben wird.

2 Historische Bemerkungen

Die Geschichte des Computers kann hier natürlich nicht auch nur im Ansatz erzählt werden, daher nur ein paar Bemerkungen und Literaturverweise. Rechenmaschinen, die dem Menschen das stumpfsinnige Rechnen erleichtern sollten, gibt es schon lange. Schon im 17. Jahrhundert wurden mechanische Rechenmaschinen konstruiert, die addieren und subtrahieren konnten und auch das Multiplizieren und Dividieren stark erleichterten [1]. Als einer der Urväter solcher Maschinen gilt Wilhelm Schickard (1592-1635). Solche mechanischen Rechenmaschinen, wurden bis in die 1970er Jahre hinaus gebaut.

Neben einfachen Rechenoperationen war es lange der Traum von Mathematikern und Ingenieuren, auch komplexe Rechenabläufe mit vielen Rechenschritten zu automatisieren. Einer der Pioniere dieser Entwicklung war Charles Babbage (1791-1871), der im 19. Jahrhundert verschiedene Maschinen konstruierte, die aber mit der damaligen Feinmechanik nicht zu realisieren waren (erst nach 1990 herum konnte man seine Maschine nachbauen) [2].

Ebenfalls mechanische Probleme hatte der deutsche Ingenieur Konrad Zuse (1910-1985) mit seinem mechanischen Rechner Z1. Bei seinem berühmt gewordenen Rechner Z3 aus dem Jahr 1941, der als erste funktionsfähige Digitalrechner gilt, wurden daher elektromechanische Relais verwendet, bei denen Magnetspulen Mehrfachschalter betätigen [3]. Damit lassen sich komplexe Strukturen allein durch Verkabelung realisieren. Ein Nachbau der Z3 steht im deutschen Museum München, in [3] findet sich auch ein Link zu einer Simulation eines Teils der Maschine. Waren die Relais ein gewaltiger Fortschritt gegenüber der reinen Mechanik, so konnte man wegen der darin enthaltenen mechanischen Schalter doch keine schnell arbeitenden Computer aufbauen (der Z3 hatte eine Geschwindigkeit von 5,3 Hz, also 5,3 Takten pro Sekunde).

Eine sehr viel höhere Geschwindigkeit konnte man mit Elektronenröhren erreichen. Der ENIAC (Electronic Numerical Integrator and Computer) aus dem Jahr 1946 bestand (unter anderem) aus mehr als 17000 solcher Röhren und wog 27 Tonnen [4]. Im Vergleich zu den 5,3Hz des elektro-mechanischen Zuse Z3 zeigen die 10kHz=10000 Hz des ENIAC den gewaltigen Fortschritt dieser elektronischen Technologie [5]. Die wesentliche informatische Struktur auch der heutigen Rechner, die in diesem Skript beschrieben wird, ist nach dem ungarisch-amerikanischen Mathematiker John von Neumann (1903-1957) benannt und stammt letztlich aus dieser Röhren-Zeit der Computer.

Das Problem der Elektronenröhren waren einerseits ihre Störanfälligkeit und andererseits ihr Platzbedarf und extremer Energieverbrauch (die Elektronen werden in den Röhren durch glühende Drähte freigesetzt). Außer für große Forschungseinrichtungen, Industrieunternehmen und das Militär waren solche Rechner praktisch unbezahlbar.

Die Erfindung des Transistors, eines elektronischen Bauteils auf Halbleiterbasis, ermöglichte in den Folgejahren, immer kleinere und leistungsfähigere Rechner zu bauen. Man spricht in diesem Zusammenhang auch von „Miniaturisierung“. Mit der damit einhergehenden Verbilligung von Rechenleistung wurden Computer immer mehr Anwendern zugänglich. Eine gute Darstellung dieser Entwicklung bietet Paul Ceruzzis Buch: The History of Modern Computing [6].

In den 1980er Jahren eroberten die Computer (Home Computer und Personal Computer genannt) die Privathaushalte. Heute haben in den Industrieländern die meisten Menschen Zugang zu Computern und nutzen dies geschäftlich und privat. Die Miniaturisierung hat in den letzten Jahren angehalten: Immer kleinere Geräte verfügen heute über erstaunliche Rechenleistung, so dass zum Teil Computer selbst dort Anwendung finden, wo wir sie gar nicht mehr bemerken. Mikrocontroller werden zur Steuerung von Abläufen in vielen elektrischen und elektronischen Geräten verwendet, vor allem auch in der Automobil-Branche, ohne, dass wir diese Geräte als Computer überhaupt wahrnehmen.

3 Aufbau eines heutigen Personalcomputers (Stand 2010)

Schraubt man einen heutigen PC auf, ergibt sich etwa das Bild aus Abbildung 1. Oben links befindet sich das Stromversorgungsgerät, auf der rechten Seite erkennt man die Schächte, in der etwa Festplatten und DVD-Laufwerke montiert werden können. Den wichtigste und kompliziertesten Teil sieht man etwa in der Mitte links des Gehäuses: die Hauptplatine (Motherboard) mit dem Prozessor (CPU). Ein Ausschnitt einer solchen Platine ist nochmal in Abbildung 2 gezeigt.

Abbildung 1: Älterer Personalcomputer

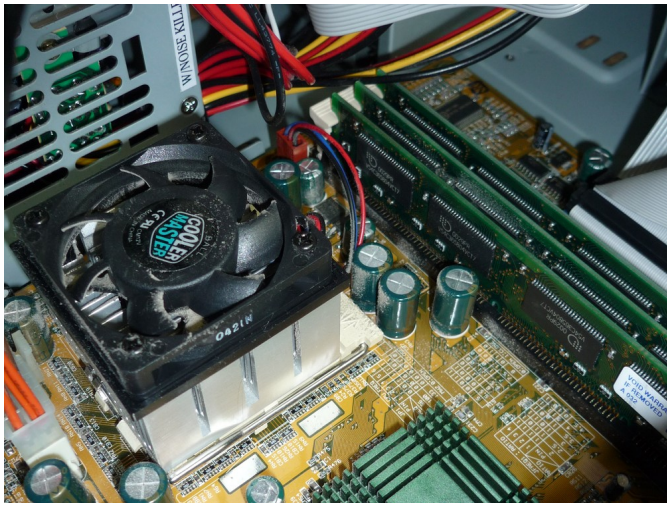


Abbildung 2: Ausschnitt der Hauptplatine (Motherboard) mit luftgekühlter CPU (links) und RAM-Speicher (rechts)

In der technischen Beschreibung von Rechnern (und natürlich auch der Werbung) wird vor allem auf den Prozessor (CPU, Central Processing Unit) und auf den Arbeitsspeicher (Memory; genauer: RAM, Random Access Memory) hingewiesen. Alle anderen Bestandteile (Festplattenspeicher, Ein- und Ausgabemedien wie USB-Schnittstelle, Tastatur, Maus, Graphik- und Soundkarte usw.) wollen wir hier als Peripherie (also „Umgebung“) bezeichnen.

RAM und Peripherie sind über ein so genanntes Bus-System mit der CPU verbunden (Das Wort Bus leitet sich aus dem lateinischen „*omnibus*“ - „für alle“ ab. Wie beim Straßen-Omnibus wird auch beim Computer-Bus das Transportsystem von verschiedenen Daten als „Verkehrsteilnehmern“ verwendet, obwohl sie unter Umständen unterschiedliche Start- und Zielpunkte haben).

Der Prozessor selbst besteht vor allem aus zwei Einheiten: dem so genannten Rechenwerk (englisch: ALU, Arithmetic Logic Unit), das mathematische und logische Operationen ausführen kann und dem Steuerwerk (englisch: Control Unit), das letztlich alle Abläufe des Rechners steuert.

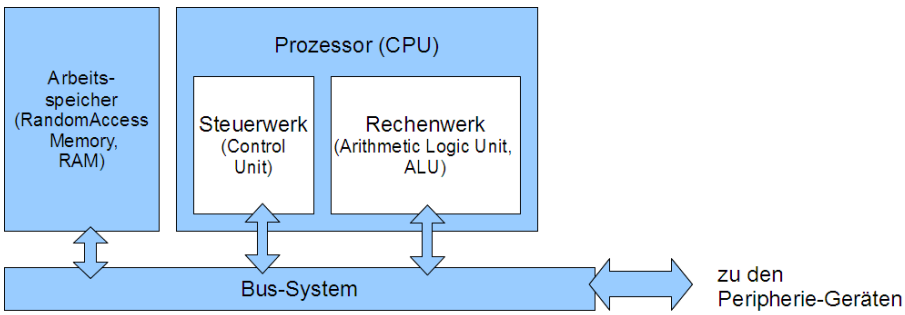


Abbildung 3: Vereinfachte Struktur eines Rechners

4 Der Johnny-Simulator

4.1 Ein vereinfachtes Rechnermodell

Wie schon im Vorwort gesagt, sieht man einem modernen Rechner seine Funktion nicht an. Um besser zu verstehen, wie Rechner funktionieren, greifen wir deshalb auf eine Simulation zurück. Der Simulator „Johnny“ ist nach dem Entwickler der heutigen Rechnerstruktur, John von Neumann (1903-1957) benannt, der privat für seine Partys berühmt und dessen Spitzname wohl auch deshalb „Good Time Johnny“ war [7].

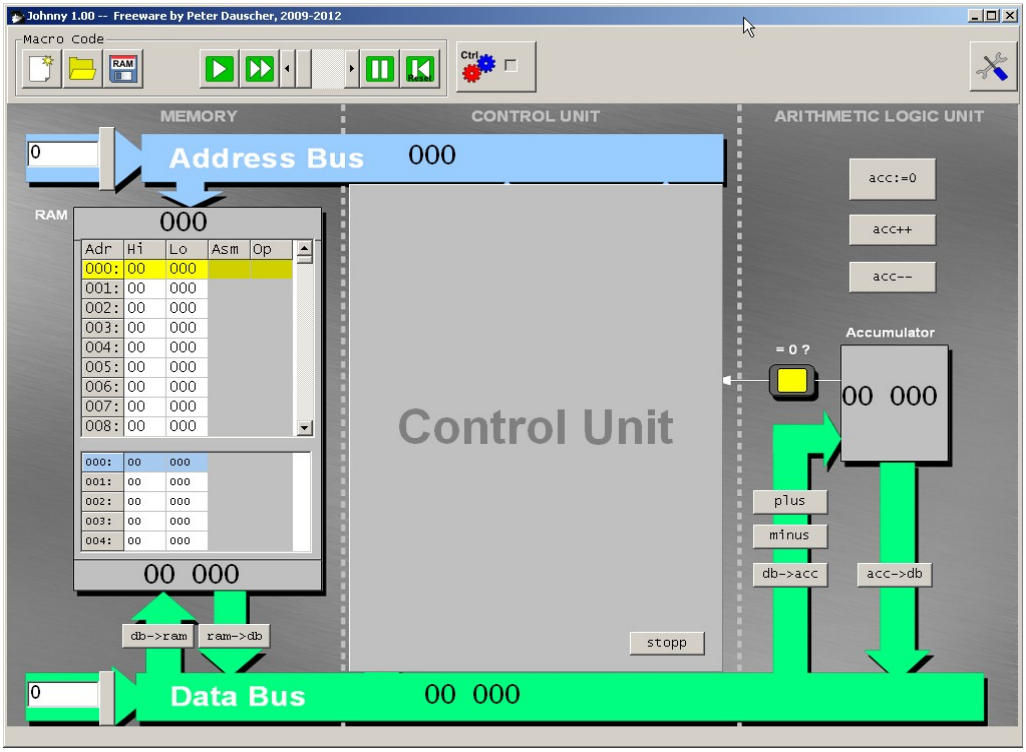


Abbildung 4: Oberfläche des Johnny-Simulatorsmit verdecktem Steuerwerk (Control Unit)

Der Johnny-Simulator konzentriert sich auf CPU mit Rechenwerk und Steuerwerk und den Arbeitsspeicher. Die in Abbildung 4 gezeigte Benutzungsoberfläche entspricht dabei etwa der Struktur in Abbildung 3:

Beim Start ist die Struktur des Steuerwerks (Control Unit) noch verdeckt, um die Dinge zunächst einfach zu halten. Der Inhalt des RAM kann im Simulator gelöscht, aus einer Datei geladen bzw. in eine Datei geschrieben werden.

Vieles ist beim Johnny-Simulator einfacher als bei einem richtigen Rechner. So rechnen praktisch alle „echten“ Rechner im Binärsystem, während der Simulator in unserem gewohnten Dezimalsystem rechnet. Auch die Funktion des Bus-Systems ist gegenüber echten Rechnern vereinfacht: Normalerweise können sich Bus-Syteme keine Daten merken, sondern sind lediglich Verbindungswege.

All diese Vereinfachungen sollen den Benutzern des Simulators die Bedienung vereinfachen, die Grundprinzipien des Rechners bleiben jedoch erhalten. Der Simulator Johnny ist Open Source Software: Er kann ohne Kosten via Internet heruntergeladen werden und auch der Programmtext selbst (in Freepascal / Lazarus geschrieben) ist frei zugänglich:

`http://sourceforge.net/projects/johnnysimulator/`

4.2 Zahlen im Arbeitsspeicher (RAM)

Betrachten wir zunächst den Arbeitsspeicher. Dieser ist im Simulator als eine Tabelle dargestellt, was seiner tatsächlichen Struktur recht nahe kommt. Jede Zeile der Tabelle hat eine Nummer, die so genannte Adresse (Spalte „Adr“). In jeder Zeile kann eine Zahl zwischen 0 und 19999 abgelegt werden. Aus technischen Gründen sind dabei die 10000er- und die 1000er-Stelle in einer gesonderten Spalte „Hi“, die 100er-, die 10er- und die 1er-Stelle „Lo“ abgelegt. Die hinteren beiden Spalten „Asm“ und „Op“ werden später eine Rolle spielen, sie beinhalten aber im Grunde nur Kommentare, die bei Zahlen auftauchen, die größer oder gleich 1000 sind. Der eigentliche RAM-Inhalt sind tatsächlich nur die Spalten „Hi“ und „Lo“.

Dies kann man (im Simulator) entweder durch Anklicken der entsprechenden Zeile, oder aber über das Bus-System des Simulators.

Zunächst muss die richtige Zeile, also Adresse gewählt werden. Dies tut man, indem man die Adresse der betreffenden Zeile am so genannten Adressbus oben links im Textfeld eingibt, mit dem nebenstehenden Knopf diese Zahl auf den Adressbus legt. Danach legt man analog die Zahl, die in den RAM geschrieben werden soll, auf den Datenbus und drückt auf den Knopf `db->ram`.

Umgekehrt kann man auch Inhalte aus dem RAM wieder auf den Datenbus legen. Man wählt zunächst die jeweilige Adresse und drückt dann auf den Knopf `ram->db`.

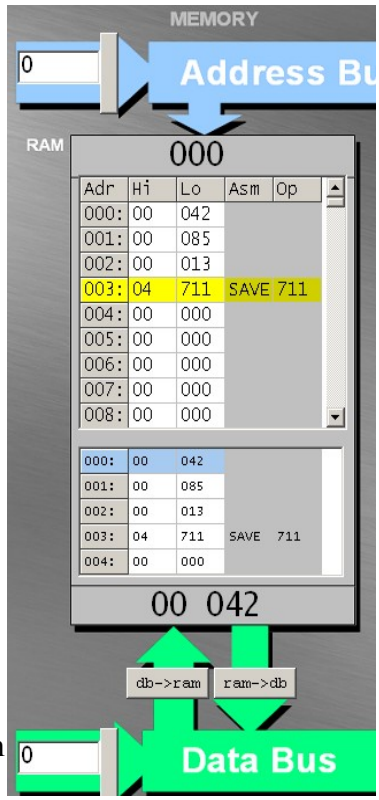


Abbildung 5: Arbeitsspeicher und seine Verbindungen zu Adress- und Datenbus

4.3 Rechnen mit Johnny

Betrachten wir nun das Rechenwerk (Arithmetic Logic Unit). Diese kann mit Daten, die auf dem Datenbus liegen, rechnen. Kernstück des Rechenwerks ist der so genannte Akkumulator (lat.: „Sammler“). In ihm sammeln sich die Rechenergebnisse an. Drückt man z.B. auf die Taste `plus`, so wird der Wert auf dem Datenbus zum Wert des Akkumulators addiert und das Ergebnis wieder im Akkumulator gespeichert. Johnny beherrscht von sich aus nur die beiden Rechenoperationen `plus` und `minus`, alles andere muss man ihm sozusagen beibringen (mehr dazu später).

Der Wert des Akkumulators kann auch wieder auf den Datenbus kopiert werden (mit `acc->db`). Daneben gibt es noch ein paar andere Möglichkeiten, den Akkumulator zu manipulieren, die Sie aber auch leicht selbst herausfinden können.

Wichtig für später ist auch die Kontrolllampe links neben dem Akkumulator. Diese zeigt an, ob der Inhalt des Akkumulators gerade den Wert 0 hat.

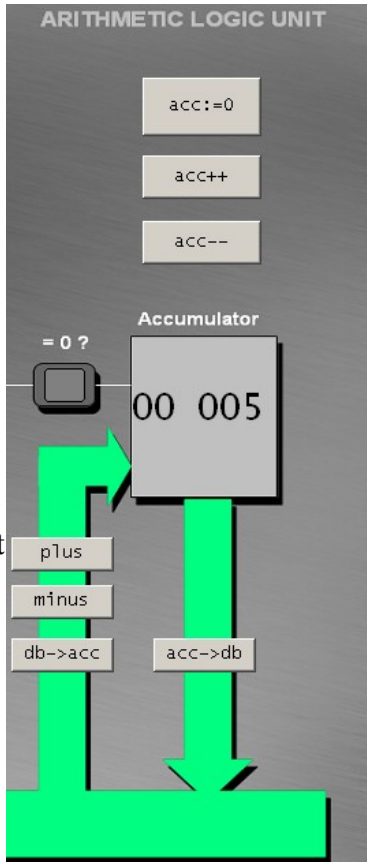


Abbildung 6: Rechenwerk von mit Verbindung zum Datenbus

5 Mikrobefehle im Handbetrieb

Bis hierher hat Ihnen das Skript nur Dinge beschrieben. Vieles jedoch können Sie auch selbst herausfinden.

Arbeitsauftrag 1

a)
Die Knöpfe des Simulators entsprechen Aktionen, die man als „Mikrobefehle“ bezeichnet; sie sind die kleinsten Befehlseinheiten, mit denen der Prozessor gesteuert werden kann.

Probieren Sie die folgenden Mikrobefehle aus und beschreiben Sie die Funktion dieser Mikrobefehle:

`db->ram`	
`ram->db`	
`acc:=0`	
`acc++`	
`acc--`	
`plus`	
`minus`	
`acc->db`	

b)
Angenommen, die Adresse „42“ würde bereits auf dem Adressbus stehen. Welche Mikrobefehle muss man nacheinander ausführen, um den Wert in der Speicherstelle mit der Adresse 42 zu verdoppeln (unabhängig davon, welche Zahl vorher gerade im Akkumulator der ALU steht).
Zum Testen sollte man vorher die Speicherstelle 42 mit einem Wert >0 belegen.

6 Makrobefehle als praktische Abkürzung

6.1 Wiederkehrende Befehlsfolgen

Arbeitsauftrag 2

Beschreiben Sie, mit welcher Folge von Aktionen man mit dem Simulator die Zahlen in den Speicherstellen 10,11 und 12 addieren und dann die Zahl in Speicherstelle 14 davon subtrahieren kann. Das Ergebnis soll in der Speicherstelle mit der Adresse 15 abgespeichert werden.
Neben den schon bekannten Mikrobefehlen brauchen wir zusätzlich noch eine Anweisung wie
„`Lege Adressse ... auf den Adressbus`“
wobei „...“ für die entsprechende Adresse, also die Nummer der Zeile im RAM steht.

Jemand behauptet, man könnte geschickte Abkürzungen für einzelne Teile dieser Befehlsfolge finden, so dass sich das Programm abgekürzt so liest:

```
TAKE    10
ADD     11
ADD     12
SUB     14
SAVE    15
```

Man nennt ein solches Programm auch Maschinenprogramm, die einzelnen Befehle darin Makrobefehle, da sie mehrere Mikrobefehle zu größeren Einheiten zusammenfassen. Gliedern Sie Ihre obige Befehlsfolge in die entsprechenden Teile und ordnen Sie ihnen in der rechten Spalte die passenden Makrobefehle aus dem Maschinenprogramm in der unteren Tabelle zu.

6.2 Der Speicher als Programmspeicher

Am Arbeitsauftrag 2 merkt man: Maschinenprograme aus Makrobefehlen lassen sich leichter verstehen und vor allem viel kompakter hinschreiben als eine Folge von Mikrobefehlen. Die Frage ist nur: wo genau hinschreiben. In den ersten Rechnern wie dem Zuse Z3 waren die Programme z.B. auf Lochstreifen gespeichert. John von Neumann hatte die Idee, nicht nur Zahlen sondern auch Programme in ein und demselben Arbeitsspeicher abzulegen [8],[9].

Wie aber lassen sich Makrobefehle im Speicher (RAM) ablegen? Wir sehen, dass der RAM nur Zahlen – in unserem Simulator zwischen 0 und 19999 – ablegen kann.

Wie also kann man einen Befehl wie `ADD 12` im RAM speichern?

Zunächst überlegt man sich hierfür, dass dieser Befehl eigentlich zweigeteilt ist, einmal in die eigentliche Operation „ADD“ und dann in die Adresse, auf die sich der Befehl bezieht. Der Speicher unseres Simulationsrechners enthält 1000 Speicherstellen mit Adressen zwischen 0 und 999. Um die Adressen darzustellen brauchen wir also die 1er-, die 10er- und die 100er-Stelle einer Zahl. Die verbliebenen zwei Stellen, die 1000er- und die 10000er-Stelle können wir verwenden, um die Operation selbst zu beschreiben, nicht mit einer Zeichenfolge wie `Add` sondern mit einer Zahl, etwa 02.000 (wir schreiben – wie im RAM – alle Zahlen mit 5 Stellen und trennen durch einen Punkt die beiden höheren Stellen ab).

Damit wird aus dem Befehl `ADD 12` nun einfach die Zahl „02.012“. Damit man sich als Benutzer des Simulators die Zahlen für die einzelnen Operationen nicht merken muss, kann man bei der Eingabe von Werten in den RAM ein Auswahlmenü anklicken, dass direkt die einzelnen Operationen mit ihren entsprechenden Zahlen zur Auswahl stellt.

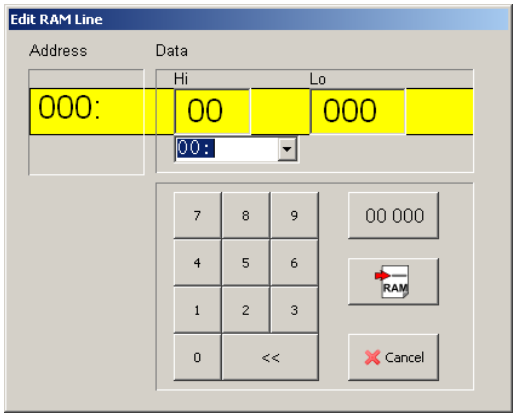


Abbildung 7: Auswahlfenster für Makrobefehle im Johnny-Simulator

Auf diese Weise kann man ohne Auswendiglernen der entsprechenden Zahlen (der so genannten Operation Codes, kurz Opcodes genannt) ein Maschinenprogramm eingeben. Neben dem eigentlichen Speicherinhalt, der – wie wir wissen – nur aus den Zahlen besteht, zeigt der Simulator als Kommentar die zugehörigen Makrobefehle an.

Adr	Hi	Lo	Asm	Op
000:	01	010	TAKE	010
001:	02	011	ADD	011
002:	02	012	ADD	012
003:	03	014	SUB	014
004:	04	015	SAVE	015
005:	00	000		
006:	00	000		
007:	00	000		
008:	00	000		

Abbildung 8: Maschinenprogramm im Speicher

6.3 Erste Maschinenprogramme

Arbeitsauftrag 3

a)
Geben Sie das folgende Maschinenprogramm ab der Adresse 000 ein:

```
000: TAKE   010
001: ADD    011
002: ADD    012
003: SUB    014
004: SAVE   015
```

Testen Sie das Programm, in dem sie unter den verschiedenen Adressen 10,11,12 und 14 verschiedene Zahlen vor dem Programmlauf speichern.

Mit der Taste  können Sie schrittweise durch Ihr Programm hindurchgehen.

Mit  können Sie wieder bei Speicherstelle 000 mit der Programmausführung beginnen. Diese Taste sollten Sie vor jedem neuen Durchlauf drücken.

b)
Neben den Befehlen `TAKE`, `ADD`, `SUB` und `SAVE` gibt es noch einige andere. Finden Sie anhand des folgenden Programms heraus, was die darin enthaltenen Befehle bewirken und geben Sie eine kurze Beschreibung in der Tabelle darunter.

Tipp: Mit Taste  können Sie den Inhalt des gesamten Arbeitsspeichers löschen.

```
000: INC    010
001: INC    010
002: DEC    010
003: INC    010
004: INC    010
005: NULL   010
```

`INC` <*Adresse*>	
`DEC` <Adresse>	
`NULL` <Adresse>	

6.4 Sprünge und Bedingungen

Bisher sind unsere Programme immer von einem Befehl zum nächsten gegangen. Für kompliziertere Aufgaben reicht das jedoch nicht. Wie Sie wahrscheinlich aus anderen Programmiersprachen wissen, muss es auch Verzweigungen (`if ... then ... else ...`) oder Schleifen (etwa `while ... do ...`) geben können.

Arbeitsauftrag 4

a)
Testen Sie das folgende Programm und finden Sie heraus, was die Befehle JMP und TST bewirken. Setzen Sie vor Ausführung des Programms die Speicherstelle Nr. 10 auf den Wert 5 und die Speicherstelle Nr. 11 auf den Wert 42.

```
000: INC   010
001: DEC   010
002: TST   010
003: JMP   001
004: TAKE  011
005: SAVE  012
```

Beschreiben Sie die Funktion der beiden Befehle JMP und TST in kurzer Form:

JMP <*Adresse*>	
TST	

b)
Nun wird es komplizierter: Konstruieren Sie ein Maschinenprogramm, das die Zahlen in den Speicherstellen 10 und 11 miteinander multipliziert und das Ergebnis in der Speicherstelle 12 abspeichert.

Tipp:
Das Programm soll folgendes tun:

- Die Speicherstelle 12 zunächst auf 0 setzen.
- Den Wert der Speicherstelle 12 in dem Akkumulator holen, den Wert der Speicherstelle 10 addieren das Ergebnis und wieder in Speicherzelle 12 abspeichern.
- Nach einer solchen Addition den Wert der Speicherstelle 11 um 1 erniedrigen.
- Das Verfahren ab dem 2. Punkt so lange wiederholen Sie, bis die Speicherstelle 11 den Wert 0 hat.

Testen Sie Ihr Programm, ob es auch funktioniert. Hierfür sollten die Speicherstellen 10 und 11 auf kleine, von Null verschiedene Werte gesetzt werden, also z.B. 5 und 3.

6.5 Programme, die sich selbst ändern

Wie bereits gesagt, ist das besondere an von Neumanns Idee zur Speicherung von Programmen, dass Programmcode („Was soll gemacht werden?“) und Programmdaten („Womit soll etwas gemacht werden?“) in ein und dem selben Arbeitsspeicher gespeichert sind.

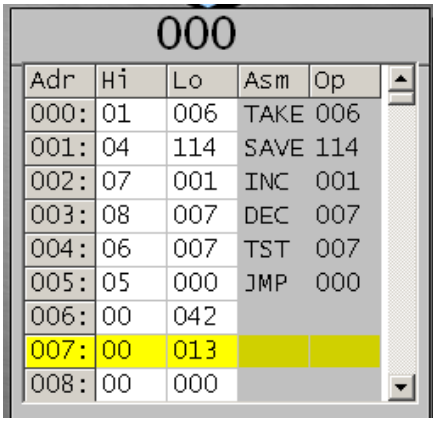
Mehr noch: Das Programm besteht selbst aus Daten, der einzige Unterschied ist, dass diese Daten, die einem Programm entsprechen, von der Maschine als Befehle interpretiert und ausgeführt werden.

Dies hat eine wichtige Konsequenz, die zunächst ein wenig abwegig klingt: Programme können sich selbst (oder andere Programme im Arbeitsspeicher) verändern, indem sie die entsprechenden Daten im Speicher manipulieren. Compiler z.B. lesen einen Text in einer höheren Programmiersprache und übersetzen ihn in ein Maschinenprogramm. So etwas geht in dieser Weise nur dank der von-Neumann-Architektur.

Einen Compiler zu bauen würde den Rahmen dieses Kurses sprengen, allerdings wollen wir uns ein Programm anschauen, das tatsächlich sich selbst, also seinen eigenen Programmcode ändert:

Arbeitsauftrag 5

a)
Untersuchen Sie das folgende Programm und beschreiben Sie seine Funktion:



b)
Modifizieren Sie das Programm so, dass es nicht ständig ein und die selbe Zahl schreibt, sondern fortlaufende Zahlen, also z.B. 42, 43, 44, 45 usw. Außerdem soll das Programm diese Zahlen nur in die ungeraden Speicherstellen ab 100 schreiben, also in die 101, die 103 usw.

7 Vom Makrobefehl zu den Mikrobefehlen – das Steuerwerk

7.1 Der Aufbau des Steuerwerks

Bisher haben wir einfach akzeptiert, dass unser Computer die Befehle im Speicher liest und die entsprechenden Mikrobefehle irgendwie ausführt. Wie aber macht er das? Schließlich handelt es sich nur um ein elektronisches Gerät ohne eigene Intelligenz (auch wenn er zugegebenermaßen intelligent konstruiert ist).

Um zu sehen, wie das Interpretieren der Makrobefehle und die Übersetzung in Mikrobefehls-Folgen geschieht, müssen wir zunächst das (bislang versteckte) Steuerwerk betrachten.

Hierfür drücken wir auf die Taste 

Das bisher verdeckte Steuerwerk wird eingeblendet und man erkennt, dass zur Steuerung eines Computers doch noch einige kompliziertere Bestandteile gehören:

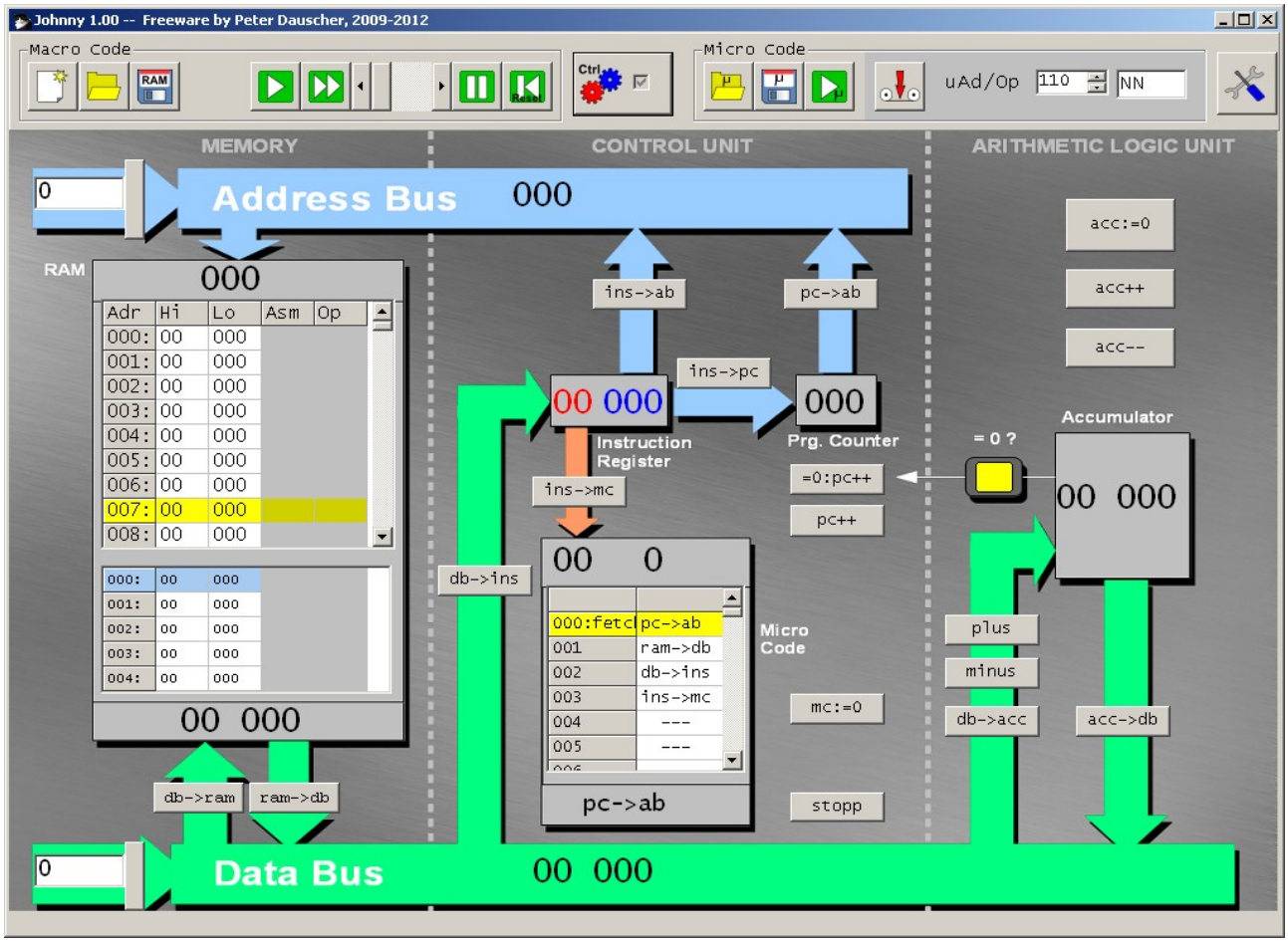


Abbildung 9: Oberfläche des Johnny-Simulatorsmit angezeigtem Steuerwerk (Control Unit)

Wichtig sind hierbei vor allem das so genannte Befehlsregister (Instruction Register), der Programmzähler (Program Counter) und der so genannte Mikrobefehlsspeicher (Micro Code). Diese sind über Leitungen miteinander verbunden und für den Transport von Daten zwischen diesen Bestandteilen und zu den beiden Bussen braucht man noch ein paar weitere Mikrobefehle.

7.2 Die Mikrobefehle des Steuerwerks

Arbeitsauftrag 6

a)
Legen Sie die Zahl 04.142 (also Befehl `SAVE 142`) auf den Datenbus.
Führen Sie dann (in der hier gegebenen Reihenfolge) die folgenden Mikrobefehle aus und notieren Sie, was dabei jeweils geschieht:

`db->ins`	
`ins->mc`	
`ins->ab`	
`pc++`	
`pc->ab`	
`ins->pc`	
`mc:=0`	

b)
Testen Sie den Mikrobefehl `=0:pc++`, einmal, wenn der Akkumulator den Wert 0 hat, einmal, wenn sie einen von 0 verschiedenen Wert hat.

c)
Was geschieht im Bereich „Micro Code, wenn Sie die Taste  wiederholt drücken?

7.3 Der von Neumann-Zyklus

In Arbeitsauftrag 6 haben Sie gesehen, dass ein im Befehlsregister (Instruction Register) stehender Makrobefehl dafür sorgt, dass der Rechner an die richtige Stelle im Mikrobefehlsspeicher (Micro Code) springt und die dort stehenden zu diesem Makrobefehl gehörenden Mikrobefehle ab der Einsprungstelle hintereinander ausführt.

Wenn die Liste der zugehörenden Mikrobefehle abgearbeitet ist, muss der nächste Befehl ins Befehlsregister (Instruction Register) geladen werden. Dies erfolgt ebenfalls über die Mikrobefehle, die Sie in Arbeitsauftrag 6 kennen gelernt haben.

Die Folge von Mikrobefehlen, die zum Laden des Befehls ins Befehlsregister notwendig ist, steht im Mikrobefehlsspeicher ab der Speicherstelle 000 und werden im Johnny-Simulator mit „fetch“ bezeichnet.

Arbeitsauftrag 7

a)
Erläutern Sie, wofür die einzelnen Befehle des fetch-Bereichs des Mikrobefehlsspeichers notwendig sind:

`pc->ab`	
`ram->db`	
`db->ins`	
`ins->mc`	

b)
Die Mikrobefehls-Folge für jeden Makrobefehl wie `TAKE`, `ADD`, `SUB` usw. endet mit den beiden Mikrobefehlen `pc++` und `mc:=0`.

Wozu dienen diese beiden Befehle?

Das abwechselnde Laden des nächsten Befehls und das Ausführen dieses Befehls wird auch als „von-Neumann-Zyklus“ bezeichnet:

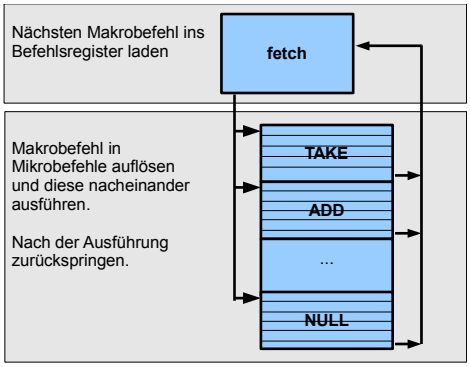


Abbildung 10: von-Neumann-Zyklus

8 Literatur

[1] http://de.wikipedia.org/wiki/Rechenmaschine

[2] http://de.wikipedia.org/wiki/Charles_Babbage

[3] http://de.wikipedia.org/wiki/Zuse_Z3

[4] http://de.wikipedia.org/wiki/ENIAC

[5] Manfred Precht u.a.: EDV-Grundwissen: Eine Einführung in Theorie und Praxis der modernen EDV. Addison-Wesley, 2004.

[6] Paul Ceruzzi: A History of Modern Computing. MIT Press, 2003

[7] http://de.wikipedia.org/wiki/John_von_Neumann

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

[9] John von Neumann: First Draft of a Report on the EDVAC, 1945