

Johnny 1.00

Version vom 10.07.2012

Simulation eines vereinfachten
von-Neumann-Rechners

Peter Dauscher, 2009-2012



– Dokumentation –



Inhaltsverzeichnis

1. Einleitung.....	3
2. Didaktische Überlegungen und Vereinfachungen.....	3
3. Der simulierte Prozessor.....	4
3.1. Speicherwerk (Memory).....	5
3.2. Rechenwerk (Arithmetic Logic Unit).....	5
3.3. Steuerwerk (Control Unit).....	6
4. Der Standard-Assembler.....	7
5. Einfache Assemblerbeispiele.....	8
5.1. Addieren zweier Zahlen und Abspeichern des Ergebnisses.....	8
5.2. Multiplizieren zweier Zahlen und Abspeichern des Ergebnisses.....	8
6. Die Benutzungsoberfläche.....	9
7. Der von-Neumannsche Befehlszyklus.....	11
8. Erweiterung des Befehlssatzes durch Modifikation des Mikrobefehlspeichers.....	12
9. Der Bonsai-Modus.....	13
9.1. Mikro- und Makrobefehle im Bonsai-Modus.....	13
9.2. Umschalten in den Bonsai-Modus.....	13
9.3. Öffnen und Speichern.....	13
10. Didaktisch relevante Änderungen in Version 1.00.....	14
11. Rechtliches, Technisches und Dank	14
11.1. Rechtliches	14
11.2. Technisches.....	15
11.3. Dank.....	16

1. Einleitung

Die von Neumannsche Struktur ist noch immer die Grundlage der meisten gebräuchlichen Rechner, wenn auch moderne Prozessoren aus Gründen der Optimierung in vielen Details von der Grundstruktur abweichen. Wie praktisch immer in der Mikroelektronik geschehen die zentralen Abläufe im Verborgenen und für Schüler, Auszubildende und Studenten sind diese damit schwer nachvollziehbar.

Der Simulator *Johnny* simuliert einen sehr einfachen Prozessor, den es als Hardware-Lösung (noch) nicht gibt und visualisiert die inneren Abläufe auf verschiedenen Abstraktionsebenen. Der Prozessor selbst wurde vornehmlich unter didaktischen und methodischen Gesichtspunkten entwickelt.

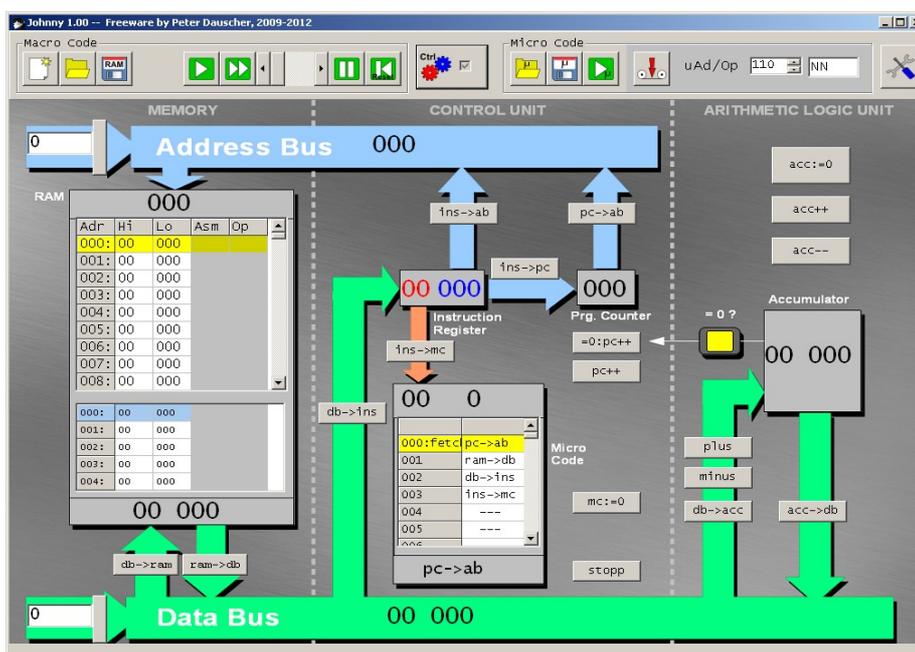
2. Didaktische Überlegungen und Vereinfachungen

- Der Simulator kann sowohl auf der Ebene der Macrobefehle als auch auf der Ebene der Microbefehle betrachtet werden. Das etwas komplexere Steuerwerk kann am Anfang der Betrachtungen ausgeblendet werden.
- Schüler können selbst Programme als Folge von Assembler-artigen Befehlen eingeben. Die Maschinenbefehle lassen sich durch Klicken erstellen und nur die Parameter müssen über die Tastatur eingegeben werden. Syntaxfehler werden auf diese Weise ausgeschlossen. Eine echte Assemblerschicht unter Verwendung von Labels, die erst später durch absolute Adressen ersetzt werden, wurde absichtlich weggelassen.
- Der Prozessor beschränkt sich auf ein einziges Register im Rechenwerk, das als Akkumulator wirkt. Zahlenwerte aus dem Speicher werden direkt zu dem Wert im Register addiert oder davon subtrahiert. Diese Einfachheit geht auf Kosten der Möglichkeit, auf den Speicher indiziert zuzugreifen, sorgt aber für ein hohes Maß an Übersichtlichkeit und einen einfachen Micro-Code.
- Der Simulator basiert auf dem Dezimalsystem. Dies ist zwar in Bezug auf die typische Hardware-Realisierung eines solchen Prozessors unrealistisch, denn natürlich arbeiten Mikroprozessoren als digitale Schaltungen im Binärsystem. Für Anfänger die Rechenabläufe jedoch viel leichter verständlich, wenn sie sich im gewöhnlichen Dezimalsystem abspielen und nicht im für die Maschinen leichter „verdaulichen“ Binärsystem oder im Kompromiss des Hexadezimalsystems. In diesem Simulator ist also $09+09$ tatsächlich 18 und nicht 12.
- Überläufe werden nicht zugelassen: $0-1=0$ und $19999+1=19999$.
- Der mitgelieferte Microbefehlsspeicher enthält einen überschaubaren Befehlssatz von 10 Assemblerbefehlen, mit denen man bereits eine große Menge typischer Probleme auf Maschinenebene lösen kann. Die Befehle sind so konstruiert, dass sie jeweils mit nur einem Argument auskommen. Die Struktur des Prozessors ist so aufgebaut, dass jeweils ein Assemblerbefehl in einer Zeile des Speichers abgelegt werden kann. Auf diese Weise wird das Holen eines Befehls relativ einfach. Die mitgelieferten Befehle beschränken sich auf eine absolute Adressierung.

- Bei realen Prozessoren besteht ein Microbefehl aus dem gleichzeitigen Aktivieren mehrerer Steuerleitungen. Diese Gleichzeitigkeit ist zum Teil nicht leicht zu durchschauen und kann bei unsachgemäßer Microprogrammierung zu Buskonflikten führen. Aus diesem Grund wurde hier eine didaktische Reduktion vorgenommen: Jeder Microbefehl entspricht genau einem Taster, den die Schüler auch manuell betätigen können. Auf diese Weise können die Schüler auch per Hand die Funktionsweise von Microbefehlen nachvollziehen.
- Ein Macro-Befehl besteht aus rein sequentiellen Abfolge solcher einfach nachzuvollziehender Steuerbefehle. Diese Einfachheit wird erkaufte durch die Tatsache, dass den beiden Bussen die Eigenschaften von Registern, also eine Merkfähigkeit zugeschrieben wird, die Busse in realen Prozessoren nicht haben. Wird also in einem realen Prozessor der Bus als reiner Transportweg verwendet und Sender und Empfänger gleichzeitig aktiviert, so wird im Simulator zunächst der Sender aktiviert, der das Daten- oder Adresswort „auf den Bus legt“, wonach im nächsten Schritt der Empfänger sich das Wort „vom Bus nimmt“. Diese Vereinfachung erscheint insofern als gerechtfertigt, da es sich ohnehin auf der einen Seite um ein stark vereinfachtes System handelt, auf der anderen Seite große Übersichtlichkeit geschaffen wird und Buskonflikte komplett ausgeschlossen sind.
- Der Microbefehlsspeicher selbst ist editierbar. Macrobefehle lassen sich von den Schülern aus Microbefehlen einfach „zusammenzuklicken“; der Inhalt des Microbefehlsspeichers lässt sich abspeichern und wieder laden.

3. Der simulierte Prozessor

Der Prozessor besteht aus drei Hauptteilen: dem Rechenwerk (Arithmetic Logic Unit), dem Speicherwerk (Memory) und dem Steuerwerk (Control Unit). Verbunden werden sie durch einen Datenbus für Werte von 0 bis 19999 und einen Adressbus für Adressen von 0 bis 999.



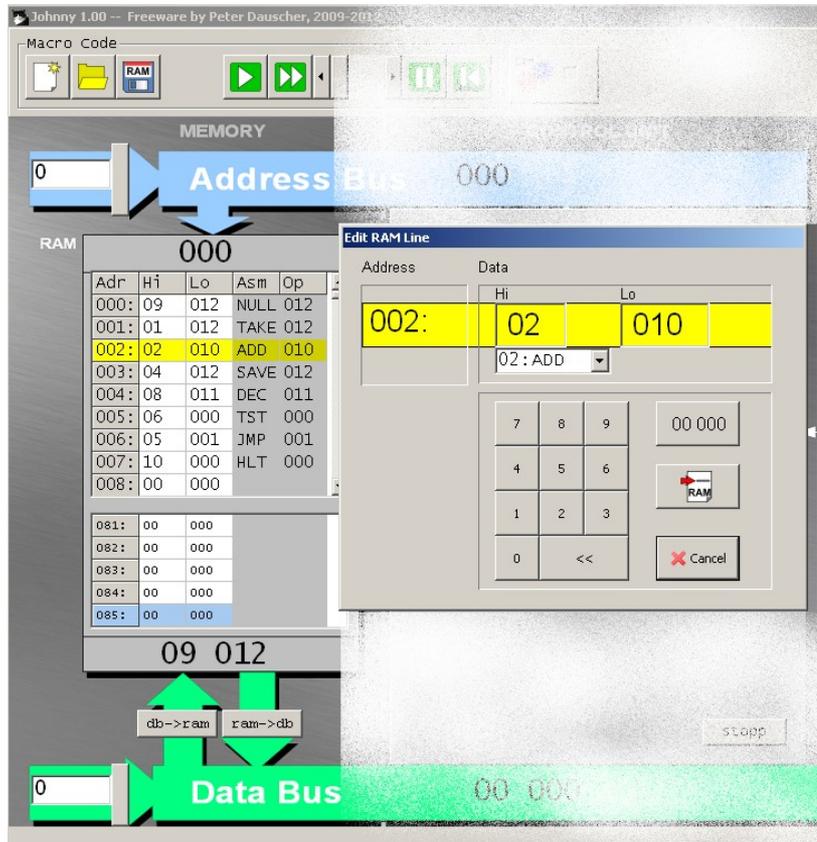
Die Komponenten sollen hier nun gesondert betrachtet werden.

3.1. Speicherwerk (Memory)

Der RAM des Speicherwerks beinhaltet 1000 Speicherzeilen, die über den Adressbus über drei Dezimalstellen adressiert werden. Jede Speicherzelle enthält Zahlen zwischen 0 und 19999; die 1000er- und 1000er-Stelle sind absichtlich optisch abgetrennt, weil in diesen Stellen die gegebenenfalls Maschinenbefehle abgelegt sind.

Mit zwei Tasten bzw. Microbefehlen (ram->db, db->ram) kann der Inhalt der derzeit adressierten Speicherstelle auf den Datenbus gelegt werden bzw.

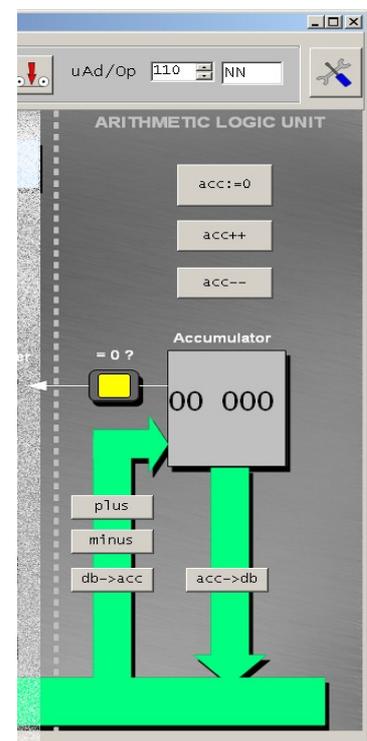
der Wert vom Datenbus in die derzeit adressierte Speicherstelle hineingeschrieben werden. Durch Klicken auf eine Speicherzelle ist diese editierbar; ihr Zahlenwert oder – wegen des Komforts – auch der entsprechende Assemblerbefehl kann direkt eingegeben werden.



Beim RAM werden zwei Ausschnitte gezeigt, die sich auch überlappen können. Der Sinn dieser Darstellung liegt darin, beim Abarbeiten gleichzeitig das abzuarbeitende Programm als auch die Bereiche des Speichers zu sehen, auf denen das Programm operiert. Die Simulationsumgebung sorgt automatisch dafür, dass der zur Zeit bearbeitete Befehl im oberen Bereich zu sehen ist, während die Speicherzellen, auf die das Programm zugreift, im unteren Teil gezeigt werden.

3.2. Rechenwerk (Arithmetic Logic Unit)

Das Rechenwerk besteht im Wesentlichen aus dem Akkumulator. Im Standard-Johnny-Modus kann man mit den Tasten (bzw. Microbefehlen) den Akkumulator auf 0 zurücksetzen (acc:=0), Werte vom Datenbus ins Rechenwerk zu transportieren (db->acc), hinzuaddieren (plus) bzw. subtrahieren (minus) und den Wert des Akkumulators auf den Datenbus legen (acc->db). Weiterhin kann man mit den Befehlen acc++ und acc-- den Wert 1 zum Akkumulator-Inhalt addieren oder davon subtrahieren.



Beim später gesondert Bonsai-Modus stehen nur noch die Befehle `db->acc`, `acc++` und `acc--` zur Verfügung, was gesondert in Abschnitt 9 beschrieben wird.

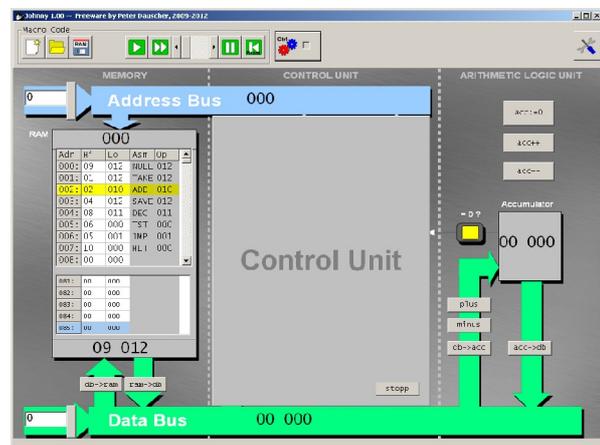
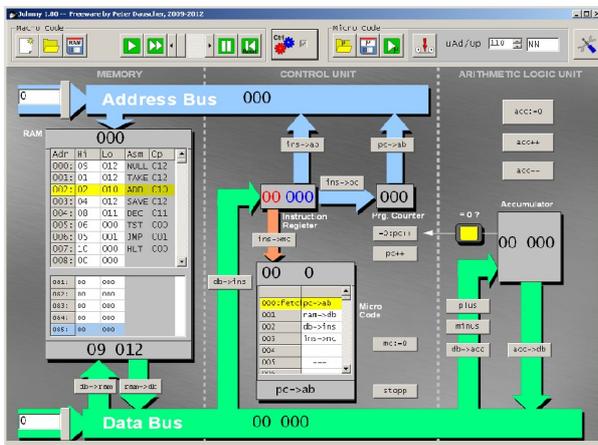
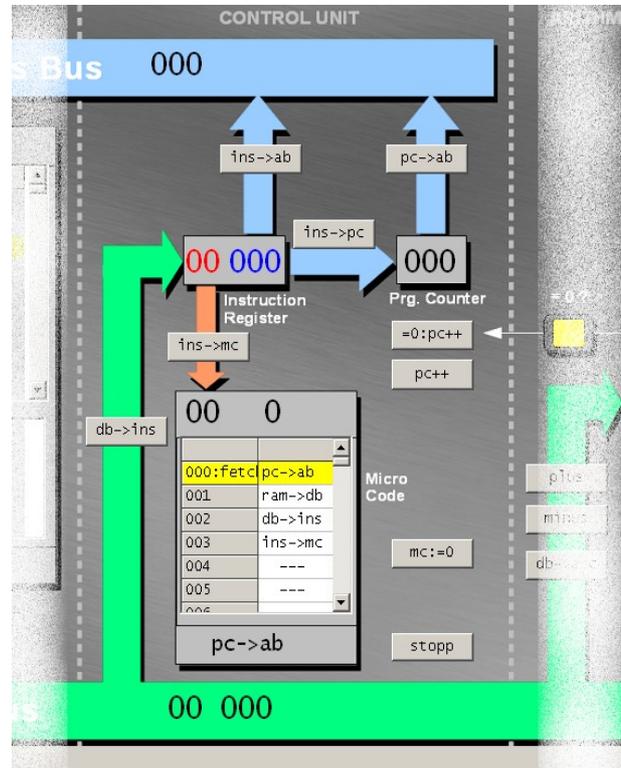
3.3. Steuerwerk (Control Unit)

Der komplexeste Teil des virtuellen Prozessors ist das Steuerwerk. Es besteht aus einem Befehlsregister (Instruction Register), dem Programmzähler und dem Microbefehlsspeicher. Durch Tastendrücke bzw. Microbefehle kann der Inhalt des Befehlsregisters vom Datenbus geholt werden (`db->ins`), der Adressteil entweder direkt auf den Adressbus (`ins->ab`) oder auf den Programmzähler (`ins->pc`) gelegt werden, oder der Wert des Programmzählers kann auf den Adressbus gelegt werden (`pc->ab`). Mit weiteren Tasten bzw. Microbefehlen kann der Programmzähler um 1 erhöht werden (`pc++`) oder bedingt um 1 erhöht werden, wenn der Akkumulator den Wert 0 hat (`=0:pc++`). Weiterhin kann mit einem Tastendruck bzw. Microbefehl die höchste Dezimalstelle des Befehlsregisters verwendet werden, um die höchste Dezimalstelle des Microbefehlsschalters zu setzen (`ins->mc`).

Die niedrigste Stelle des Microbefehlsschalters wird dabei automatisch auf 0 gesetzt. Schließlich und letztlich gibt es eine Taste bzw. einen Microbefehl, der den Microbefehlsschalter, der den gesamten Microbefehlsschalter auf 0 setzt (`mc:=0`).

Der Microbefehl `stopp` bewirkt nichts an der eigentlichen Steuerung, sondern gibt lediglich eine Meldung aus, dass das Programm abgearbeitet ist.

Wie bereits erwähnt, kann man das Steuerwerk ein- und ausblenden, wobei es sich empfiehlt, es zunächst auszublenden, um mehr Übersichtlichkeit zu gewinnen.



4. Der Standard-Assembler

Der mitgelieferte Microbefehlsspeicher speichert die Steuerung von 10 Makrobefehlen ab:

- **TAKE** Der Inhalt einer absolut adressierte Speicherzelle wird in den Akkumulator der Arithmetisch-Logischen Einheit geladen.
- **SAVE** Der Inhalt des Akkumulators wird in in eine absolut adressierte Speicherzelle geladen.
- **ADD** Der Inhalt einer absolut adressierte Speicherzelle wird zum Wert im Akkumulator der Arithmetisch-Logischen Einheit addiert.
- **SUB** Der Inhalt einer absolut adressierte Speicherzelle wird vom Wert im Akkumulator der Arithmetisch-Logischen Einheit subtrahiert.
- **TST** Hat die angegebene Speicherstelle den Wert 0, so wird bei der Ausführung des Programms eine Speicherstelle übersprungen.
(Das ist eine Änderung zu früheren Versionen von Johnny. Dort wurde der gerade im Akkumulator befindliche Wert getestet. Die Änderung erfolgte aus Gründen der Einheitlichkeit). 
- **JMP** Das Programm wird an der absolut angegebenen Speicheradresse fortgesetzt.
- **INC** Der Inhalt einer absolut adressierten Speicherzelle wird um 1 erhöht.
- **DEC** Der Inhalt einer absolut adressierten Speicherzelle wird um 1 erniedrigt.
- **NULL** Der Inhalt einer absolut adressierten Speicherzelle wird auf 0 gesetzt.
- **HLT** Es wird eine Meldung angezeigt, dass das Programm abgearbeitet ist. Läuft der Simulator im Dauerbetrieb, wird der Lauf unterbrochen.

Durch Editieren des Microbefehlsspeichers ist es möglich, selbstständig bis zu zehn weiten Macrobefehle aus Microbefehlen zu konstruieren. Eine genaue Erläuterung folgt in Abschnitt 8.

Seit Version 0.98 steht auch der noch stärker reduzierte Befehlssatz des Bonsai-Rechners zur Verfügung, was gesondert in Abschnitt 9 beschrieben wird.

5. Einfache Assemblerbeispiele

5.1. Addieren zweier Zahlen und Abspeichern des Ergebnisses

Soll die Zahl aus Speicherzelle 10 zur Zahl aus Speicherzelle 11 addiert und das Ergebnis in Speicherzelle 12 gespeichert werden, so lässt sich dies durch das folgende einfache Maschinenprogramm realisieren:

```
001:    TAKE    010
002:    ADD     011
003:    SAVE    012
004:    HLT     000
```

5.2. Multiplizieren zweier Zahlen und Abspeichern des Ergebnisses

Hier muss man nun zwei Speicherstellen mit Hilfwerten belegen: Zum Beispiel Speicherstelle 9 mit 1 und die Speicherstelle 12, in der nachher das Ergebnis stehen soll, mit 0.

```
000:    NULL    012
001:    TAKE    012
002:    ADD     010
003:    SAVE    012
004:    DEC     011
005:    TST     011
006:    JMP     001
007:    HLT     000
```

Das Programm addiert nacheinander immer wieder den Wert in Speicherstelle 10 auf den Wert in Speicherstelle 12 auf und speichert das Ergebnis wieder in Speicherstelle 12 ab. In jedem solchen Zyklus vermindert das Programm dann den Wert in Speicherstelle 11 um 1. Das geschieht so lange, bis in Speicherstelle 11 der Wert 0 steht. Dann springt das Programm aus der Schleife heraus und hält an.

6. Die Benutzungsoberfläche

Die Bedienknöpfe der Benutzungsoberfläche sind in die Gruppen Macro Code und Micro Code unterteilt. Die Bedienknöpfe zum Micro Code sind nur bei eingblendeten Steuerwerk sichtbar, das Ein- und Ausblenden erfolgt mittels der Taste.

Die Tasten in der Macro-Code-Gruppe bedeuten:

	RAM komplett mit Zeileneinträgen 00 000 füllen
	Einlesen eines Maschinenprogramms in den RAM von der Festplatte
	Speichern des RAM-Inhalts auf der Festplatte
	Ausführen des nächsten Maschinenbefehls im RAM
	Getaktetes Ausführen des gesamten Maschinenprogramms. (Die Geschwindigkeit kann mit dem nebenstehenden Schieberegler beeinflusst werden.)
	Anhalten des getakteten Ausführens
	Zurücksetzen des Programmzählers auf Speicherstelle 0 und Löschen aller Register
	Optionenfenster zum Umschalten zwischen Johnny- und Bonsai-Modus anzeigen.

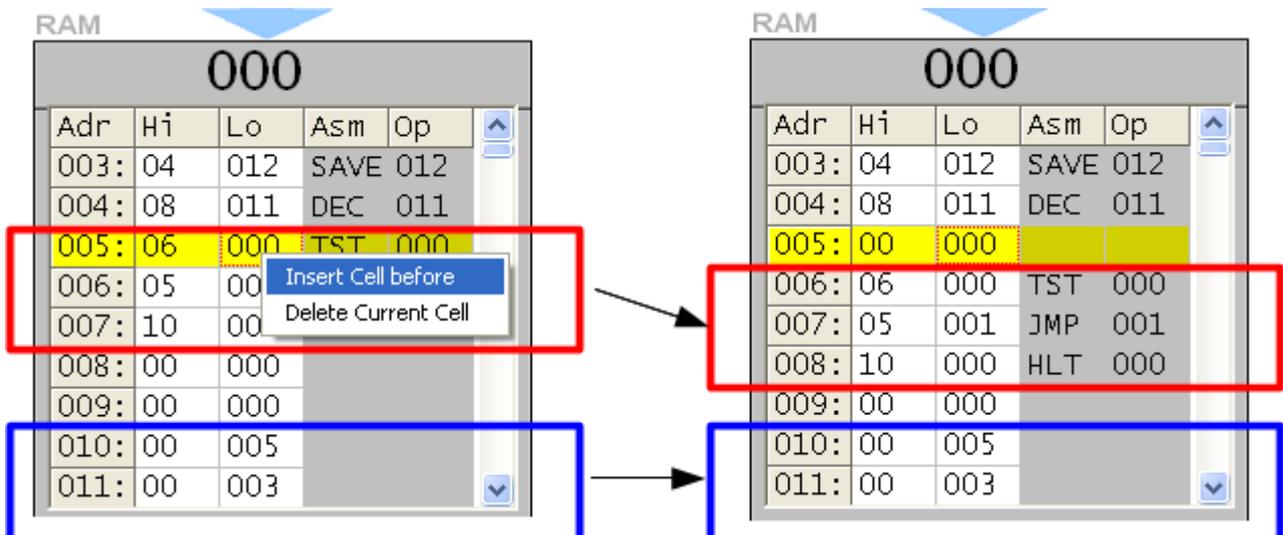
Danach kommt die Taste  zum Einblenden der Microbefehls-Abstraktionsebene. Die zugehörigen Tasten sind:

	Laden eines Microbefehlsspeicher-Inhalts von Festplatte
	Speichern des Microbefehlsspeichers auf der Festplatte
	Ausführen eines einzelnen Microbefehls
	Aufzeichnen eines Makros bestehend aus einer Sequenz von Microbefehlen. Ab der angegebenen Adresse werden die geklickten Microbefehle abgespeichert. Der angegebene Name wird als Kürzel für den Macrobefehl verwendet. Erneutes Drücken der Taste beendet die Aufzeichnung.

Klickt man in eine Speicherstelle des RAM, so erscheint ein Fenster, in dem ein Assemblerbefehl aus einem Auswahlménü ausgewählt werden kann. Der Adressteil des Befehls kann direkt über die (echte oder virtuelle) Tastatur eingegeben werden. Ein Doppelklick auf den Adressteil löscht den Inhalt. Soll die Zeile des RAM keinen Befehl sondern nur ein Datenwort darstellen, so kann auch direkt über Tastatur auf die beiden höchsten Dezimalstellen zugegriffen werden .

	Die ganze Zeile wird auf 0 gesetzt.
	Die eingegebene Zahl oder der eingegebene Befehl wird ins RAM geschrieben.
	Die Zeile bleibt unverändert.

Durch Rechtsklick auf dem RAM erscheint ein Kontextménü, über das eine Zeile im RAM eingefügt oder gelöscht werden kann.



Der Block der nachfolgenden von 00 000 verschiedenen Zeilen (im Beispiel 005-007) wird dabei jeweils um eine Zeile nach unten verschoben, eine 00000-Zeile am Ende des Blocks gelöscht. Der Rest des RAM-Inhalts (ab Zeile 010) bleibt unverändert. Bei dieser Operation ist Vorsicht geboten: Da sich die Adressen des verschobenen Teils verschieben, müssen gegebenenfalls Sprungadressen geändert werden.

Analoges geschieht beim Löschen einer Zeile. Hier rutscht der nachfolgende Block der von 00000 verschiedenen Zellen nach oben, an seinem unteren Ende wird eine 00000-Zeile eingefügt.

7. Der von-Neumannsche Befehlszyklus

Am Beispiel der Maschinenbefehlszeile `TAKE 42`, der in der 0-ten Zeile des Speichers steht, soll nun gezeigt werden, wie ein von-Neumannscher Befehlszyklus in dem virtuellen Prozessor abläuft. Hierbei werden die beiden typischen Schritte des von-Neumann-Zyklus vorgenommen, nämlich zunächst das Laden des Befehls in das Befehlsregister und dann das Abarbeiten der entsprechenden Folge von Microbefehlen.

Im Programmzähler soll bereits die Adresse des Befehls im Speicher, also 00 stehen. Beim Laden des Befehls in das Befehlsregister wird bei allen Befehlen gleichermaßen die folgende Sequenz von Microbefehlen abgearbeitet:

1. Anwählen der im Programmzähler stehenden Adresse (`pc->ab`).
2. Auslesen der entsprechenden Zeile des Speichers auf den Datenbus (`ram->db`)
3. Laden des Befehlsregisters vom Datenbus (`db->ins`). Die Nummer des Befehls "TAKE", in diesem Fall die 1, steht nun im vorderen Teil des Befehlsregisters.
4. Laden des Microbefehlszählers mit der höchsten Dezimalstelle des Befehlsregisters (`ins->mc`)

Der Microbefehlszähler steht nun auf dem Befehl `TAKE`, der nun abgearbeitet wird.

1. Zurücksetzen des Akkumulators auf den Wert 0 (`acc:=0`)
2. Anwählen der im Adressteil des Befehlsregisters stehenden Adresse, hier also 42 (`ins->ab`)
3. Auslesen der entsprechenden Zeile des Speichers auf den Datenbus (`ram->db`)
4. Addieren des Datenbus-Wertes auf den Akkumulator (`plu`)
5. Erhöhen des Befehlszählers um 1 (`pc++`)
6. Rücksetzen des Microbefehlszählers auf 0 (`mc:=0`)

Die letzten beiden Befehle dienen dazu, das Holen des nächsten Befehls aus dem Arbeitsspeicher vorzubereiten. Die anderen Befehle sind ähnlich realisiert.

8. Erweiterung des Befehlssatzes durch Modifikation des Mikrobefehlspeichers

Um den Inhalt des Mikrobefehlsspeichers zu modifizieren, also einen Macrobefehl hinzuzufügen oder zu modifizieren, kann folgendermaßen vorgegangen werden:

Zunächst wird die Adresse des Makrobefehls im Mikrobefehlsspeichers eingegeben sowie das zugehörige Kürzel (Mnemonic). Letzteres kann dann bei der späteren Verwendung des Maschinenbefehls im RAM direkt wieder aus der Auswahlliste entnommen werden.



Dann wird der Aufnahmeknopf gedrückt.

Der entsprechende Teil des Mikrobefehlsspeichers wird zunächst auf 0 gesetzt. Durch betätigen der Bedienknöpfe werden die entsprechenden Zeilen des Mikrobefehlsspeichers erstellt. Während der Aufnahme blinkt ein Teil der Benutzungsoberfläche als Warnung. Erneutes Drücken des Aufnahmeknopfes wird die Aufzeichnung des Makros beendet.

Der so modifizierte Inhalt des Mikrobefehlsspeichers kann auf Festplatte abgespeichert und wieder geladen werden.



Beim Speichern werden jeweils zwei Dateien angelegt: Eine mit dem Inhalt des Mikrobefehlsspeichers (.mpc) und eine mit den Namen der darin enthaltenen Befehle (.nam).

9. Der Bonsai-Modus

In den 1990er Jahren haben Klaus Merkert und Walter Zimmer bereits einen ähnlichen Simulator entwickelt, der mittlerweile in verschiedenen Varianten von Software-Implementationen als auch in Form einer echten Hardware-Lösung zur Verfügung steht.

Die zur Zeit verfügbaren Bonsai-Simulatoren haben ein im Vergleich zu „Johnny“ leicht unterschiedliches didaktisches Konzept; sie bilden die tatsächlichen Hardware-Strukturen und Abläufe eines Rechners typischerweise noch detailgetreuer ab und verfügen über einen nochmals deutlich reduzierten Befehlssatz von nur noch 5 Makrobefehlen.

Seit Version 0.98 kann man „Johnny“ nun in den „Bonsai-Modus“ umschalten. In diesem Modus wird der Bonsai-Befehlssatz verwendet; die zu Bonsai nicht passenden Mikrobefehle werden ausgeblendet.

9.1. Mikro- und Makrobefehle im Bonsai-Modus

Im Bonsai-Modus existieren die Mikrobefehle `plus`, `minus` und `acc:=0` nicht. Weiterhin kennt der Standard-Bonsai-Befehlssatz nur die Befehle `INC`, `DEC`, `JMP` und `TST`, sowie teilweise den Befehl `HLT`.

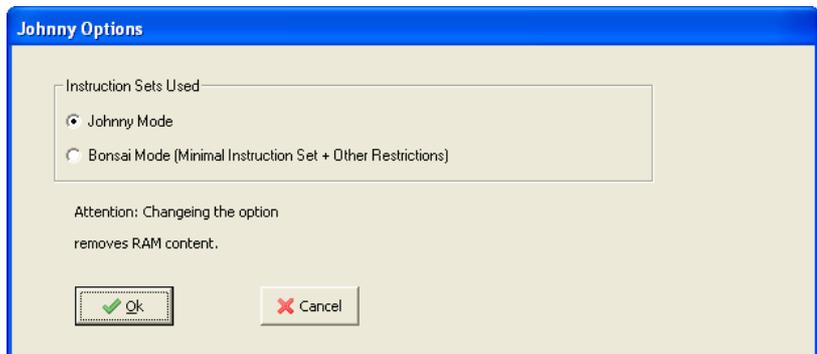
9.2. Umschalten in den Bonsai-Modus

Durch Betätigen der Taste



wird das Optionenfenster angezeigt. Hier kann man den Bonsai-Modus auswählen.

Vorsicht: Beim Ändern der Option wird der RAM-Inhalt komplett gelöscht.



9.3. Öffnen und Speichern

Bonsai-Maschinen(.bma)-Dateien des ursprünglichen Bonsai-Simulators können geöffnet und Programme auch als .bma-Dateien exportiert werden, so dass sie vom Bonsai-Simulator weiterverarbeitet werden können.

Damit keine Verwechslungen entstehen, wird beim normalen Speichern im Bonsai-Modus eine andere Endung (.bij) verwendet. Beim Laden einer solchen Datei bzw. einer .bma-Datei wird der Simulator automatisch in den Bonsai-Modus geschaltet.

Insgesamt gibt es also drei Dateiformate für RAM-Inhalte:

.ram	Standard-Johnny-RAM-Inhalte
.bma	Standard-Bonsai-Maschinen-Programme
.bij	Bonsai-Programme im Standard-Format von „Johnny“

10. Didaktisch relevante Änderungen in Version 1.00

Beim Übergang zu Version 1.00 haben sich eine Reihe von Änderungen ergeben:

- Das Befehlsregister wurde von „Command Register“ zu „Instruction Register“ umbenannt.
- Der Akkumulator wurde früher als „ALU“ bezeichnet; er heißt jetzt tatsächlich Akkumulator und die Mikrobefehle, die den Akkumulator betreffen, entsprechend auch.
- Im Bonsai-Modus werden nun Bonsai-Maschinen-(.bma)-Dateien eingelesen und keine direkten Bonsai-(.bon)-Dateien. Letztere beziehen sich auf die Harvard-artige Architektur des vereinfachten Bonsai-Simulators, was nicht mit der von-Neumannschen Struktur von Johnny verträglich ist.
- Der TST-Befehl des Standard-Assemblers wurde abgeändert. Wurde in früheren Versionen der bereits im Akkumulator vorhandene Wert getestet, wird nun Wert der Speicherstelle getestet, die im Adressteil des Befehls angegeben ist.
- Bei der Eingabe von Speicherzellen kann nun eine virtuelle Tastatur verwendet werden, so dass der Einsatz des Simulators in Kombination mit interaktiven Whiteboards erleichtert wird.

11. Rechtliches, Technisches und Dank

11.1. Rechtliches

Das Programm steht ab der Version 1.00 unter der GNU General Public License in der Version 3.

<http://www.gnu.org/licenses/gpl-3.0.txt>

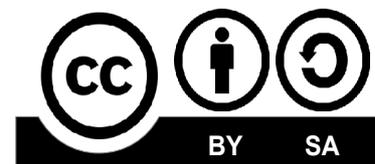


Eine Gewährleistung für die einwandfreie Funktion wird in keiner Weise gegeben; Installation und Anwendung erfolgt auf eigenes Risiko.

Ebensowenig wird technische Unterstützung in irgendeiner Weise garantiert.

Die Dokumentation selbst steht unter der Creative Commons Lizenz CC-BY-SA

<http://creativecommons.org/licenses/by-sa/3.0/de/legalcode>



11.2. Technisches

Das Programm muss nicht installiert werden; die Verfügbarkeit auf einem Datenträger (Festplatte; Netzwerplatte; USB-Stick; CD-ROM, etc.) reicht aus.

Das Programm wurde unter der freien Entwicklungsumgebung Lazarus (Version 0.9.30.4) erstellt:

<http://www.lazarus.freepascal.org>

Damit sollte sich das Programm (theoretisch) unter allen Betriebssystemen kompilieren und ausführen lassen, unter denen Lazarus in dieser oder einer kompatiblen Version verfügbar ist.

Die Graphiken des Prozessorschemas und des Splash-Screens sowie die vorliegende Dokumentation wurden unter Verwendung OpenOffice bzw. LibreOffice und dem Gnu Image Manipulation Program (GIMP) realisiert:

<http://de.openoffice.org>

<http://de.libreoffice.org>

<http://www.gimp.org>

Die Icons der Schaltflächen wurden mit Inkscape erstellt

<http://www.inkscape.org>

11.3. Dank

Allen, die durch ihren Einsatz an den oben genannten Open-Source-Projekten indirekt an diesem Simulator mitgewirkt haben, möchte ich an dieser Stelle ausdrücklich danken.

Ebenso gilt mein Dank allen, die durch Bug-Reports und Verbesserungsvorschläge geholfen haben, das Programm benutzerfreundlicher und besser zu machen. Und die mich ermutigt haben, „Johnny“ weiterzuentwickeln. Besonders möchte ich erwähnen:

- Die Mitglieder der Grundkurse Informatik (Abitur-Jahrgang 2009) des Gymnasiums am Kaiserdom in Speyer, die den Simulator als erstes getestet und wertvolle Anregungen gegeben haben.
- Meine Kollegen Herrn Jens Fiedler und Herrn Ewald Bickelmann vom Gymnasium am Kaiserdom in Speyer, sowie Herrn Bernd Fröhlich vom Nikolaus-von-Kues-Gymnasium in Bernkastel-Kues und Herrn Ernst-Lothar-Stegmaier vom Frauenlob-Gymnasium in Mainz, die Johnny in verschiedenen anderen Kursen eingesetzt und mir sehr wichtige Rückmeldungen gegeben haben.
- Herrn Alexander Güssow vom Christian-Doppler-Gymnasium Salzburg und Herrn Joachim Brehmer-Moltmann vom Gymnasium Mainz-Gonsenheim, die das Programm einer Reihe von Härtetests unterworfen haben.
- Herrn Alexander Domay vom Pamina-Gymnasium Herxheim, der das Programm ebenfalls getestet und als erster unter Linux kompiliert hat.
- Herrn Klaus Merkert vom Hohenstaufen-Gymnasium Kaiserslautern, Herrn Tobias Selinger vom Humboldt-Gymnasium in Trier für wichtige Anregungen.
- Die Organisatoren der imedia 2011 und des MNU-Bundeskongresses 2011, die ein Forum bereitgestellt haben, um „Johnny“ einer breiteren Öffentlichkeit vorzustellen.
- Alle Kollegen, die Fehler und Ungereimtheiten in der Software aufgespürt haben.
- Herrn Martin Oehler, Herrn David Meder-Marouelli und Herrn Klaus Merkert für Ihre Ermutigung, die Programmquellen des Projekts offen zu legen.
- Ganz besonders diejenigen, die ich in der Aufzählung vergessen habe.

Ich wünsche allen Benutzern viel Spaß und erfolgreiches Arbeiten mit dem „Johnny“-Simulator.

Über weitere Bug-Reports und Anregungen oder einfach nur Rückmeldungen freue ich mich und bedanke mich schon einmal vorab.

Peter Dauscher, 10.07.2012

peter.dauscher@gmail.com

