# Johnny 1.00

## Simulation of a
## Simplified von Neumann Computer

Peter Dauscher, 2009-2012

## – User's Manual –

# Content

# 1.  Remark

The author of this manual is not a native English speaker. So if you find spelling errors, grammatical or other mistakes, please feel encouraged to report them to make this user's manual better. Thank you very much in advance!

- Peter Dauscher


peter.dauscher@gmail.com



# 2.  Introduction

Most contemporary computers base upon the Von Neumann Architecture.  Due to miniaturization, most processes in modern computers are hidden to the users and can hardly be observed. Therefore, simulation is a powerful method to give pupils and students an overview about what happens in computers.

JOHNNY has been developed especially for educational purposes resulting in some substantial simplifications compared to real computers.

# 3.  Simplifications

- The simulator can be observed on the level of micro instructions as well as of macro instructions. The inner structure of the Control Unit can be masked in order to make things simpler in the beginning.

- Users can program the simulator using an assembler-like language. The respective macro instructions can be written directly into the memory using a GUI so that syntax errors are ruled out by construction.

- The Arithmetic Logic Unit consist of one single register acting as accumulator.

- The simulator and its GUI use the decimal system. Although this is not very realistic in a technical sense, it simplifies the usage especially for beginners being too familiar with hexadecimal system: So, in Johnny, 9+8 equals 16 and not 0x11.

- The data range is 0..19999; the address range is 0..999. Overflows are not allowed: 0-1 → 0 and 19999+1 → 19999.

- The instruction set is very small consisting only of 10 macro instructions. All instructions use absolute addressing (one address per instruction) for the sake of simplicity. The OP-Code is given by the tenthousands' place and the thousands' place; the other places represent the address. (ADD 42, e.g, is represented by 02 042).

- In real processors, a micro instruction activates, in general, several lines of the controlling bus. Activating more than one line in real processors, however, can easily effect bus errors if not performed properly. Therefore, a (quite unrealistic) simplification has been made: Each micro instruction corresponds to one single push button that also can be pushed manually by the user.

- A macro instruction is simply a sequence of such micro instructions. In order to avoid the necessity for simultaneity that the data bus has the ability to store a data word (which, of course, is not realistic: real buses are simply lines; storage is done by processor registers).
  Transferring a data word from one place to another via the data bus is performed in two steps: a) the sender copies the word onto the bus; b) the receiver copies the word from the bus. Bus conflicts are therefore ruled out by construction.

- Micro Code is editable: The user can create his own macro instructions simply by choosing an appropriate name and then clicking the sequence of micro instructions with the mouse.

# 4. The Processor

The processor consists of three parts: the Arithmetic Logic Unit, the Memory (RAM) and the Control Unit. These Units are interconnected by busses.
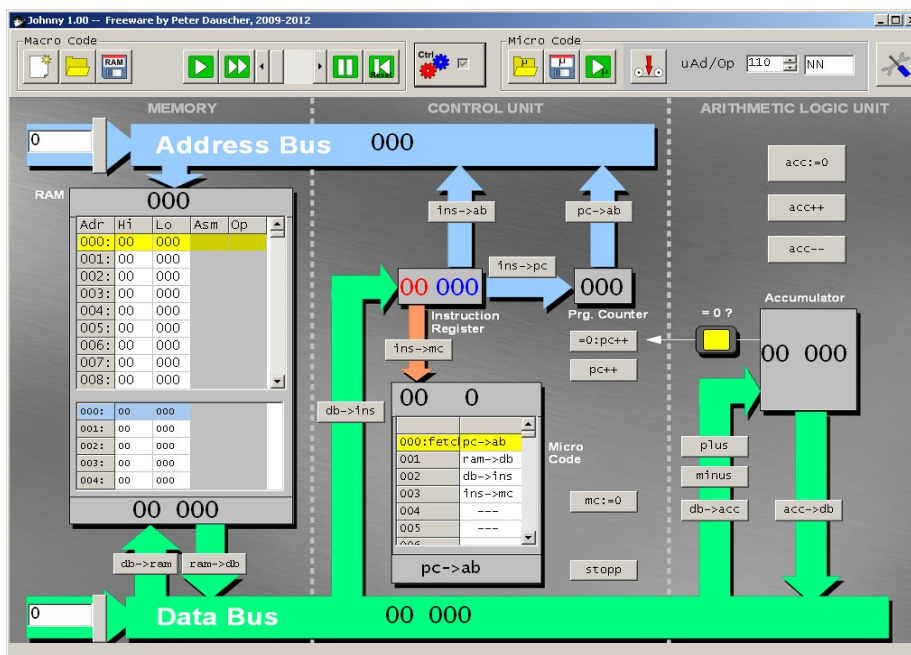


In the following each unit shall be considered in detail.

## 4.1. Memory (RAM)

The Random Access Memory (RAM) consists of 1000 locations, each having the ability to store numbers in the range 0..19999. Consequently, three decimal digits are sufficient to address each location.

The 10000s place and 1000s place are separated a bit from the other places since they represent the OP-Code of the macro instructions.

Using two push buttons (triggering the micro instructions `ram->db` and `db->ram`) data from the addressed location can be put onto or taken from the bus, respectively.
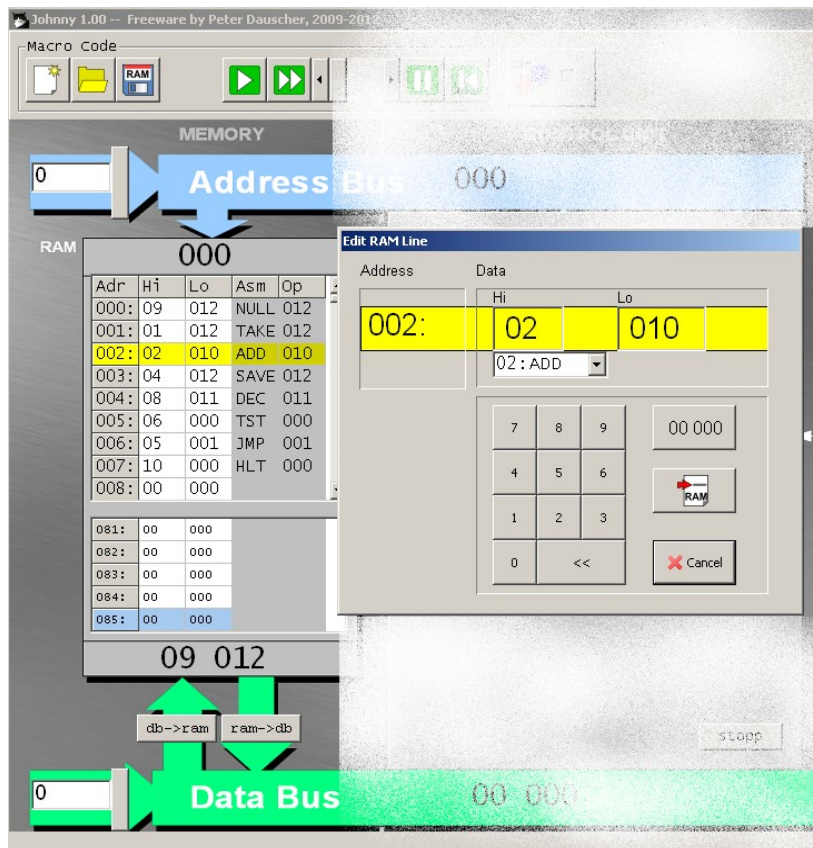


The locations are editable using the GUI; macro instructions can be chosen using a pull-down menu. In the GUI, two sections of the RAM (which can overlap) are shown. Thus, the instructions and the data affected can be shown simultaneously.

## 4.2. The Arithmetic Logic Unit

The Arithmetic Logic Unit consists merely of the accumulator. The accumulator can be resetted (`acc:=0`), incremented (`acc++`), decremented (`acc--`). `db->acc` transports a data word from the bus into the accumulator; `acc->db` does the opposite.
A value from the bus can be added (`plus`) or subtracted (`minus`).

In the so-called BONSAI Mode (to be explained later in section 9) some of these micro instructions are suppressed.
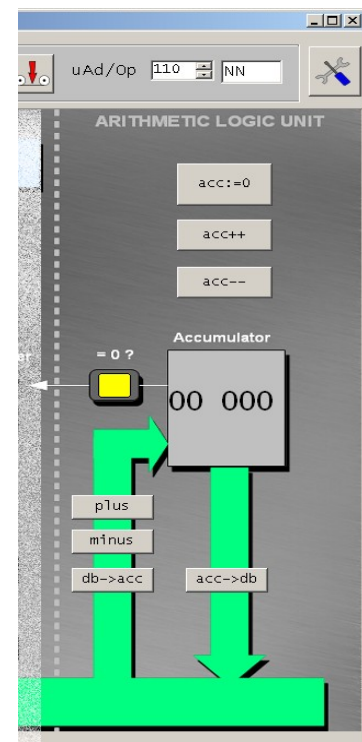
## 4.3. The Control Unit

The most complex part of the processor is the Control Unit. It consists of the instruction register, the program counter and the micro code. By `db->ins` the content of the bus is transferred to the instruction register. The address part of the instruction can be put directly onto the address bus (`ins->ab`) or transferred to the program counter (`ins->pc`), which is necessary to implement a JUMP instruction. The program counter itself can be copied to the address bus by `pc->ab`.

The micro instruction `pc++` increments the counter; `=0:pc++` does the same, but only if the accumulator has zero value.

`ins->mc` sets the 100s and 10s places of the micro instruction counter (above the micro code) to the OP-Code in the Instruction Register (and resets the 1s place to zero). The micro instruction `mc:=0` resets the micro instruction counter; `stopp` is not an instruction in a narrower sense: it only forces the simulator to show a message that the program is finished.

As mentioned above, the Control Unit can be masked in order to make things simpler.
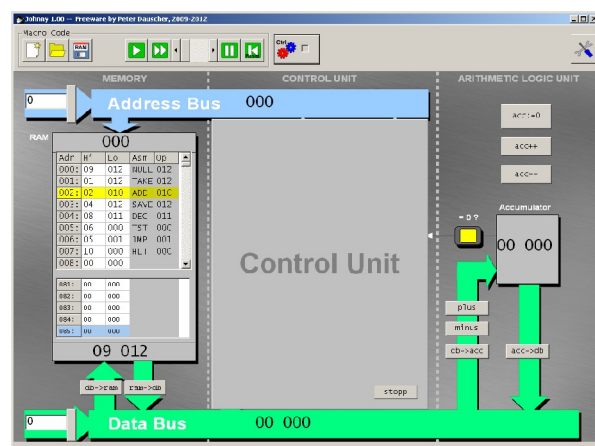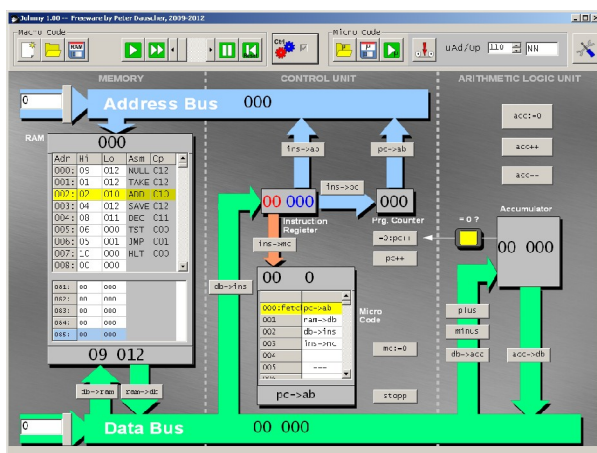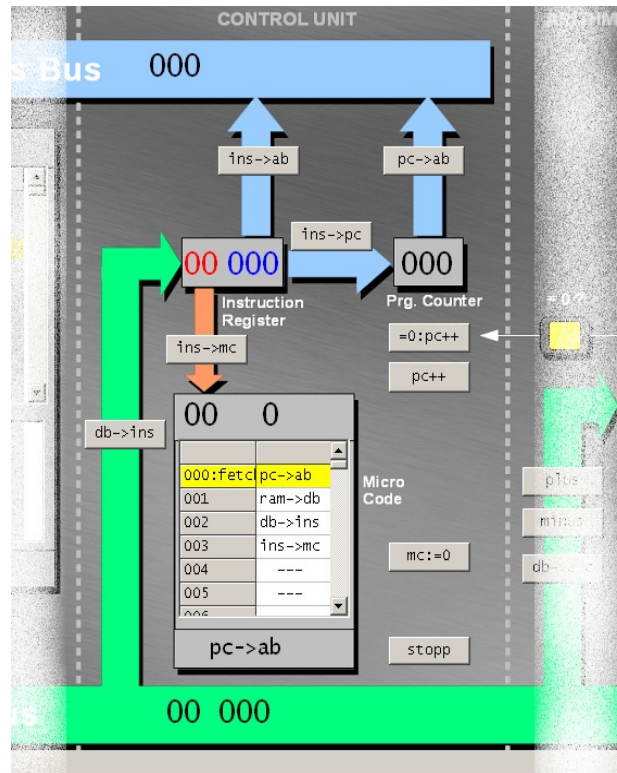
# 5. The Standard (Macro) Instruction Set

Der mitgelieferte Microbefehlsspeicher speichert die Steuerung von 10 Makrobefehlen ab:

- `TAKE`  The value of the location (given by the absolute address) is transported to the accumulator.

- `SAVE`  The value of the accumulator is transported to the location given by the absolute address.

- `ADD`    The value of a location (given by the absolute address) is added to the value in the accumulator.

- `SUB`    The value of a location (given by the absolute address) is subtracted from the value in the accumulator.

- `INC`    The value of the location (given by the absolute address) is incremented.

- `DEC`    The value of the location (given by the absolute address) is decremented.

- `NULL`  The value of the location (given by the absolute address) is set to zero.

- `TST`    If and only if the location (given by the absolute address) has a zero value, the next macro instruction is leaped over.
  (This is a substantial change compared to version 0.98 of the simulator where directly the value of the accumulator has been tested. The change has been made for the sake of uniformity.)

- `JMP`    The program is continued at the given location.

- `HLT`    The simulator shows a message that the program is finished.

# 6. Simple Program Examples

## 6.1. Adding numbers

The following example program adds the values of locations <10> and <11> and stores the result in location 12:

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

## 6.2. Multiplying numbers

Multiplication of numbers can be implemented by subsequently adding the one value <10> to the result location <12> which had set to zero, previously.

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

In each loop the other value of the other location <11> is decremented. The loop is continued until <11> has reached value 0.

# 7.  The User Interface

The buttons are arranged in two groups (Macro Code and Micro Code) the latter of which is only shown if the Control Unit is shown in detail.
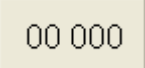
The buttons in the Macro Code group mean:

| | |
|---|---|
| | Set RAM completely to Zero (00 000 for each location) |
| | Open a program on HD. |
| | Save a program to HD. |
| | Execute the next macro instruction in the RAM. |
| | Execute the program automatically (where the speed can be controlled by the scroll bar to the right of the button). |
| | Stop the execution of the program. |
| | Set the program counter and all other processor pegisters to zero. |
| | Show the options window. |

If the button [Ctrl] is used to show the Control Unit in detail, the following buttons are shown:

| | |
|---|---|
|  | Open a micro code file on HD. |
|  | Save the current micro code to HD. |
|  | Execute a single micro instruction. |
|  | Record a micro instruction sequence to form a new macro instruction. |

If one location of the RAM is clicked, a window appears where the numerical value can be changed or a macro instruction can be chosen from a pull-down menu. The address can be entered by using the real or a virtual keyboard. A double click on the address part sets it to zero.

| | |
|---|---|
| 00 000 | The location is set to zero. |
| RAM | The change is written into the respective location. |
| ✖ Cancel | The location keeps unchanged. |

By using the right mouse button, a context menu is shown. Using this menu insertions an deletions in the RAM can be effected.
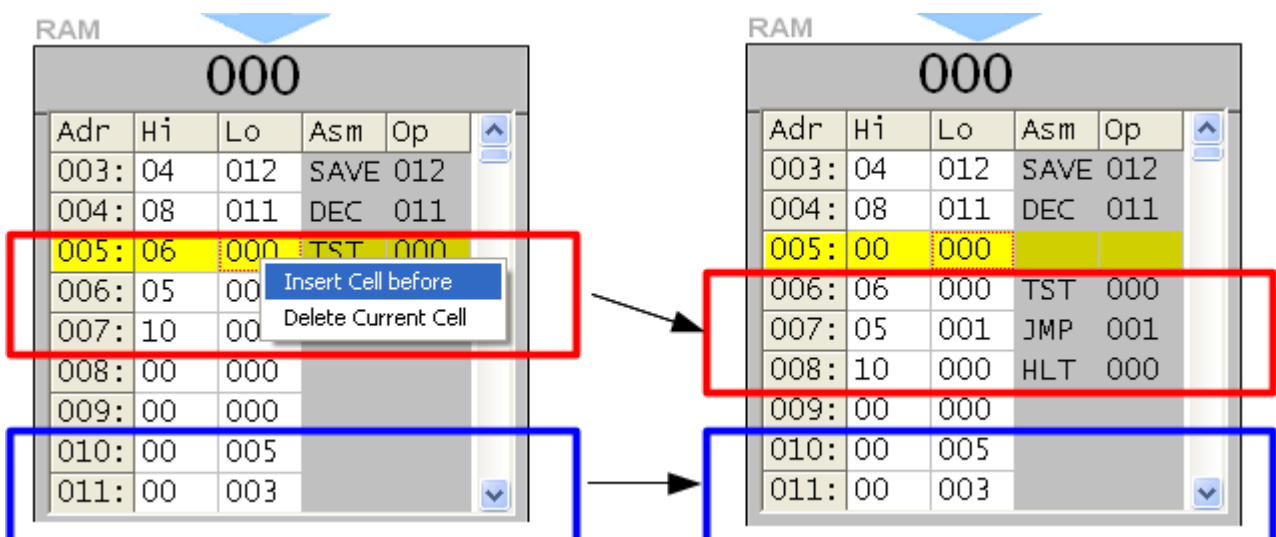
With an insertion, the block of the next non-zero-value locations is shifted down by one; one zero-value-location at the end of the block is deleted. Similarly, with a deletion the non-zero-value block is shifted up and a zero-value location is inserted below the block.
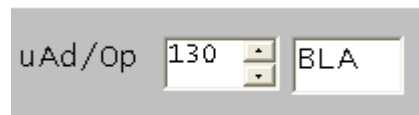
# 8.  Creating own Macro Instructions

In order to create an own macro instruction (or to change an existing) first an OP-Code and a Mnemonic must be chosen. The latter can be chosen in the pull-down menu later on when RAM content is changed.

Then the Record button is pressed:

The respective part of the micro code is set to zero at first. By pressing the buttons of the respective micro instructions, the sequence is recorded. The recording is indicated by a blinking part of the user interface. Pressing the Record button again stops the recording.

The new micro code can be saved to HD and opened again by

Saving a micro code creates two files. One (.mpc) file contains the micro code itself, the .nam file the Mnemonics of the instructions.

# 9.  The BONSAI Mode

In the 1990s Klaus Merkert and Walter Zimmer implemented a similar simulator, BONSAI. In order to be able to use the BONSAI instruction set (which consists only of five instructions, namely `INC`, `DEC`, `TST`, `JMP` and `HLT`), Johnny can be switched to BONSAI mode.

The pull-down menu is then adapted to these instructions, some micro instructions (unnecessary in this mode) are suppressed.
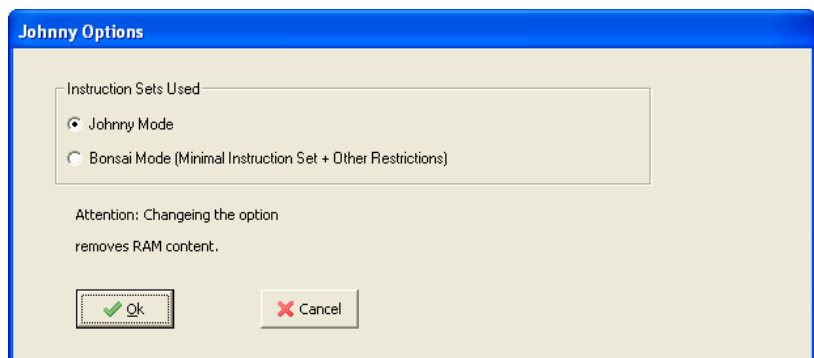
## 9.1.  Umschalten in den Bonsai-Modus

Using



a window is shown which enables the user to change the mode.

Attention: Changing the option deletes the RAM content completely.



## 9.2.  Öffnen und Speichern

BONSAI machine programs (.bma files) of the original BONSAI simulator can be opened and written. Therefore, it is possible to create a BONSAI machine program using JOHNNY and then transfer it to the original simulator (which is more complex but also more realistic).

Therefore there are three file formats for RAM content:

| .ram | Standard JOHNNY RAM content |
|------|-----------------------------|
| .bma | Standard Bonsai-Machine Programs |
| .bij | Bonsai-Programms saved in Standard JOHNNY format |

# 10.  Legal Stuff, Technical Stuff and Acknowledgements

## 10.1.  Legal Stuff

The program is Open Source and licensed under the GNU GPLv3.

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

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EX-PRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MER-CHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL PETER DAUSCHER OR ANY OTHER CONTRIBUTOR BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

The documentation itself is licensed under the Creative Commons License CC-BY-SA

http://creativecommons.org/licenses/by-sa/3.0/de/legalcode
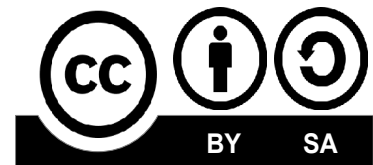
## 10.2.  Technical Stuff

The program has not to be installed in order to work: it can simply be run from any data storage medium.

The program has been developed using the free IDE Lazarus (Version 0.9.30.4):

http://www.lazarus.freepascal.org


So, at least in theory, the program should be able to be compiled to any operating system Lazarus can generate executable files for.

All graphics have been created using OpenOffice, LibreOffice, the Gnu Image Manipulation Program (GIMP) and InkScape:


http://de.openoffice.org

http://de.libreoffice.org

http://www.gimp.org

http://www.inkscape.org

## 10.3.  Acknowledgements

Thanks to all who have helped to create this simulator indirectly by their activities in the Open Source projects mentioned above.

Furthermore, I wish to thank to all persons who have given me useful feedback and bug reports:

- My students at the Gymnasium am Kaiserdom, Speyer.
- My colleagues Jens Fiedler, Ewald Bickelmann, Bernd Fröhlich, Ernst-Lothar Stegmaier and all colleagues who have tested the program with their students and found bugs.
- Alexander Güssow and Joachim Brehmer-Moltmann who tested the program extensively from the students perspective.
- Alexander Domay for the first compilation on a Linux system.
- Klaus Merkert and Tobias Selinger, Martin Oehler, David Meder-Marouelli for useful comments and the encouragement to make Johnny an Open Source project.
- The responsibles of the imedia and MNU conferences who gave me the possibility to present Johnny at a larger public.
- Especially those persons I have forgotten in my enumeration.

I wish you all much fun (and productive work) using Johnny.

Any further bug report or suggestion is appreciated. Thanks in advance!

Peter Dauscher, 10.07.2012

peter.dauscher@gmail.com