Dokumentation Pseudo - Assembler

1. Technische Daten des Von-Neumann-Rechners

- Die Anzahl der Speicherzellen im Speicherwerk des VNR ist nicht künstlich beschränkt. Aus Leistungsgründen können aber nur maximal 512 Speicherzellen angezeigt werden. Das Programm funktioniert aber auch noch, wenn Adressen über 511 (bei 0 beginnend) verwendet werden.
- Das Programm und die Daten befinden sich wie beim Von-Neumann-Prinzip im selben Speicher, weshalb zur Vermeidung von Speicherkorruption darauf geachtet werden sollte, dass das Programm während der Laufzeit nicht verändert wird, z.B. durch die Wahl von Adressen, die auf keinen Fall im Bereich des Programmcodes liegen.
- Zur besseren Übersicht und Verständlichkeit arbeitet der Pseudo-Assembler im Dezimalsystem mit ganzen Zahlen. Es besteht auch keine prinzipielle Größenbeschränkung der Zahlen.

2. Die Syntax

- **Jede Zeile** enthält genau einen Befehl. Zwischen Befehl und Operand muss mindestens ein Leerzeichen stehen. Die Befehle richten sich dabei nach dem aktuellen Befehlssatz, der verändert werden kann. Tabulatoren können als Trennzeichen ebenfalls verwendet werden. Einrückungen am Zeilenanfang sind auch möglich.

- Der gesamte Quelltext ist **case-sensitiv**, d.h. Groß- und Kleinschreibung werden unterschieden.

- Das **Ende** eines Programms kann durch den Befehl „end“, bzw. durch den Befehl, der bei „ende“ (Hilfe -> Befehlssatz anpassen) festgelegt wurde, festgelegt. (optional)

- **Adressen** werden durch ein vorangestelltes @ gekennzeichnet. (z.B. Adresse 50: @50)

- An Stelle von Adressen (adr) oder Werten (val) können auch **Variablen** verwendet werden, die dann nach ihrer Initialisierung an Stelle dieser Adressen oder Werte eingesetzt werden können. Variablennamen dürfen nur aus Buchstaben bestehen. Die Initialisierung beginnt mit dem Befehl „def“ (define/definiere). Dabei wird einer Variablen eine Speicheradresse oder ein (numerischer) Wert zugewiesen. Das sieht dann wie folgt aus: „def a = @50“ oder „def a = 3“. Auch das überschreiben von Variablen ist möglich, einfach wieder den Befehl „def“ verwenden und eine neue Adresse oder Wert zuweisen, z.B. „def a = @60“. (Beachte: Variablen müssen direkt initialisiert werden, d.h. getrennte Deklaration und Initialisierung ist nicht möglich.)

- **Sprungziele** sind Zeilennummern als normale Zahl.

3. Die Oberfläche

- Die Oberfläche ist in drei Hauptsektionen gegliedert:
  - Im Bereich „Code Editor“ wird das Pseudo-Assembler-Programm geschrieben, wobei sich die Befehle nach dem aktuellen Befehlssatz (dieser kann verändert werden) richten. Außerdem wird dort das Programm gestartet, es kann manuell beendet werden und wenn Ablauf -> VN-Phasen ausgewählt ist, muss der Button „Weiter“ gedrückt werden. Während der Laufzeit wird immer die Zeile hervorgehoben, die der Rechner gerade ausführt.
  - Im Bereich „Speicherwerk“ wird der Speicher des Von-Neumann-Rechners und das darin gespeicherte Pseudo-Assembler-Programm dargestellt. Während der Laufzeit wird immer die aktuelle Gruppe an Speicherzellen hervorgehoben, mit der der Rechner gerade arbeitet.
  - In der letzten Sektion werden die restlichen Bestandteile des Von-Neumann-Rechners dargestellt: Steuerwerk, Rechenwerk, Eingabe – sowie Ausgabewerk. Diese werden während der Laufzeit aktualisiert, sodass immer ersichtlich ist, was gerade im Rechner passiert.
- Die Menüleiste:
  - Im Reiter „Datei“ kann der Nutzer ein neues Programm erstellen, ein vorhandenes öffnen oder das aktuelle speichern. Dabei ist das Dateiformat das Textdokument (.txt), was eine unkomplizierte Verwendung und Möglichkeit zum Teilen ermöglicht. Außerdem kann unter dem Punkt „Demo“ auf verschiedene vorgefertigte Programme zurückgegriffen werden, die sich auch mit der Veränderung des Befehls-satzes verändern.
  - Im Reiter „Ablauf“ wird festgelegt wie das Programm ausgeführt wird. Unter dem Punkt „Vollständig“ wird das Programm bis zum Ende automatisch ausgeführt und es kann ausgewählt werden, wie viel Zeit für die drei VN-Phasen (Von-Neumann-Phase) Fetch–Decode–Execute gebraucht werden soll. Wird der Punkt „VN-Phasen“ ausgewählt, muss durch Drücken von „Weiter“ jede VN-Phase separat ausgeführt werden.
  - Im Reiter „Optionen“ kann der Befehlssatz angepasst werden und die Anzahl der angezeigten Speicherzellen kann eingestellt werden.
  - Im Reiter „Hilfe“ kann diese Dokumentation geöffnet werden und es können Informationen über dieses Projekt angezeigt werden.
- Shortcuts:
  - Strg + Leertaste: Start
  - Strg + N: Neu
  - Strg + O: Öffnen
  - Strg + S: Speichern
  - Strg + Alt + S: Speichern als

4. Der Standardbefehlssatz

- Der def-Befehl zur Initialisierung von Variablen gehört nicht zum Befehlssatz des Prozessors. Stattdessen werden beim Kompilieren alle Stellen, an denen eine Variable genutzt wird, durch den Wert dieser Variablen ersetzt. Deshalb ist der def-Befehl auch nicht veränderbar.
- adr = Adresse val = Wert (numerisch, ganzzahlig) tar = Sprungziel (Zeilennummer)

Befehl		Code	Funktion
ld	adr/val	101/100	**load**: Lade den Wert in adr, bzw. val in den Akku
st	adr	111	**store**: Speichere den Wert des Akku in adr
in	adr	121	**input**: Schreibe den Wert des Eingaberegisters in adr
out	adr	131	**output**: Schreibe den Wert in adr ins Ausgaberegister
add	adr/val	201/200	**add**: Addiere den Wert in adr, bzw. val zum Akku
sub	adr/val	211/210	**subtract**: Subtrahiere den Wert in adr, bzw. val zum Akku
mul	adr/val	221/220	**multiply**: Multipliziere den Wert in adr, bzw. val mit dem Akku
div	adr/val	231/230	**divide**: Dividiere den Wert in adr, bzw. val mit dem Akku
mod	adr/val	241/240	**modulo**: Rest bei Division des Akku durch den Wert in adr, bzw. val
cmp	adr/val	251/250	**compare**: Vergleiche den Akkuinhalt mit dem Wert in adr, bzw. val
jmp	tar	300	**jump**: Springe zum Ziel tar (Zeilennummer oder Marke)
jlt	tar	310	**jump if less than**: jump, wenn bei cmp der Akkuinhalt kleiner war
jeq	tar	320	**jump if equal**: jump, wenn bei cmp der Akkuinhalt gleich war
jgt	tar	330	**jump if greater than**: jump, wenn bei cmp der Akkuinhalt größer war
end		400	**end**: Beendet das Programm (optional)