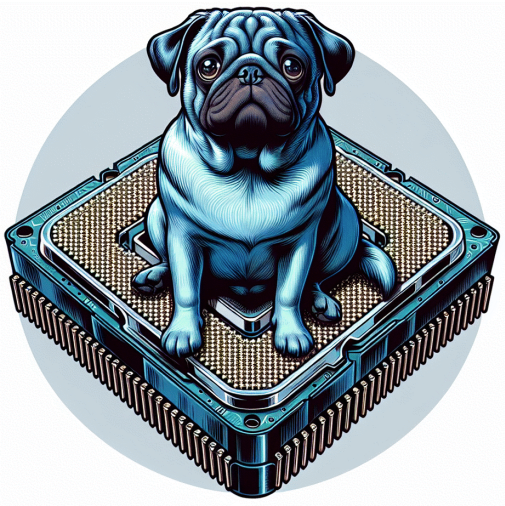
MOPS



MOdellrechner mit PSeudo-Assembler

Simulation eines von-Neumann-Rechners mit integriertem Pseudo-Assembler

Version 1.02

vom 27.03.2024

Marco Haase · *teacher@hamao.de*

Inhalt

1. Einführung....3
1.1 Was ist der MOPS?....3
1.2 Systemvoraussetzungen und Lizenz....4
1.3 Über diese Dokumentation....4

2. Grundlagen der Bedienung....4
2.1 Die integrierte Entwicklungsumgebung (IDE)....4

3. Der Assembler....5
3.1 Was der MOPS-Assembler (nicht) kann....5
3.2 Die Syntax....5
3.3 Der Befehlssatz....6

4. Der Von-Neumann-Rechner....7
4.1 Technische Daten....7
4.2 Die Ablaufsteuerung....7

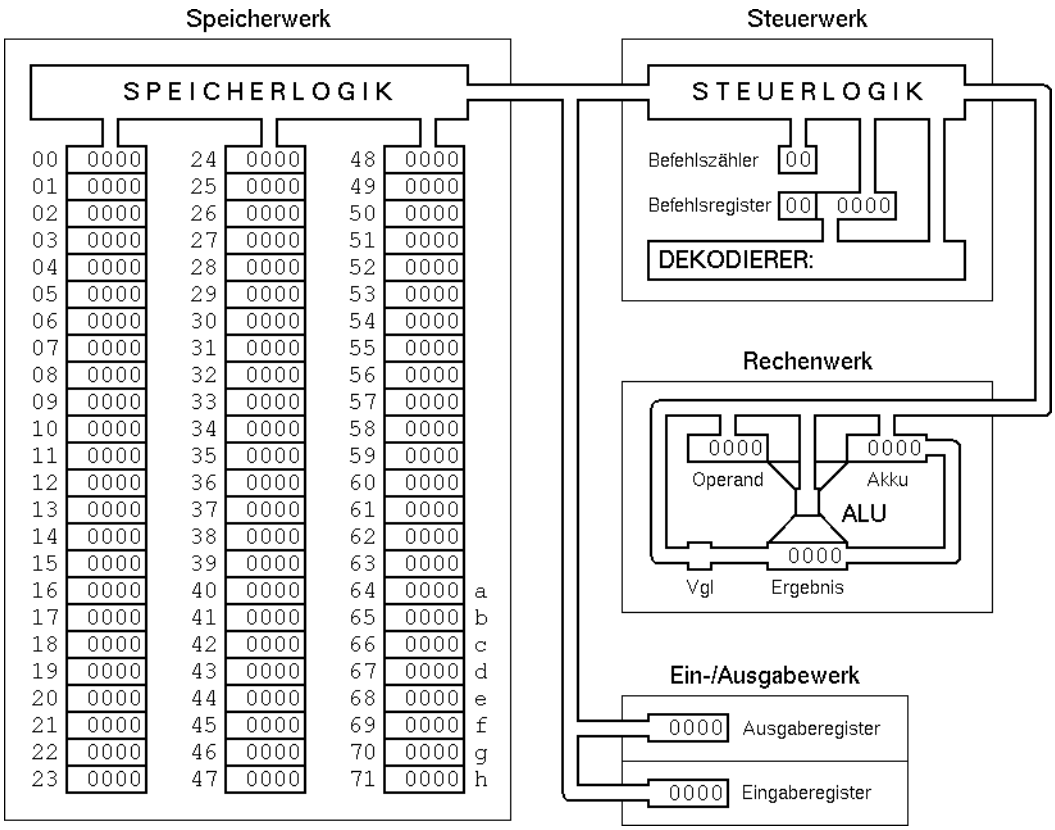
5. Beispiel....8

6. Sonstiges....9
6.1 MOPS mit einer Assemblerdatei starten....9
6.2 Fenstergröße anpassen....9

1. Einführung

1.1 Was ist der MOPS?

MOPS ist ein Modellrechner gemäß dem schematischen Aufbau eines Von-Neumann-Rechners (VNR) mit integriertem Assembler und Quelltext-Editor. Entwickelt wurde der MOPS für den Einsatz im schulischen Informatikunterricht der Sekundarstufe I.



Simuliert werden die Vorgänge, die sich beim Ablauf eines Programms im Herzen eines am VNR orientierten Rechners abspielen („Von-Neumann-Zyklen“) - siehe Abbildung. Damit man den VNR in Aktion sehen kann, muss der MOPS gefüttert werden - mit Befehlen in Maschinencode. Weil echter Maschinencode aber für Menschen nicht gut lesbar ist, enthält der MOPS außerdem noch einen Assembler, der mnemonischen Assemblercode in Maschinencode umwandelt und diesen dem VNR zuführt. Im Rahmen des Befehlsvorrats dieses Assemblers ist der MOPS vom Anwender frei programmierbar. Schließlich beinhaltet der MOPS noch einen integrierten Quelltext-Editor, damit man für das Erstellen der Assemblerquelltexte keinen separaten Editor bemühen muss.

Weil all das nur eine Simulation ist – es wird kein *echter* Maschinencode erzeugt und ausgeführt, sondern nur Pseudocode für den simulierten VNR –, ist der MOPS eben „nur“ ein **MO**dellrechner mit **PS**eudo-Assembler.

1.2 Systemvoraussetzungen und Lizenz

Die **Windows-Version 1.00** des MOPS benötigt mindestens **Windows XP**, die Version 1.02 mindestens **Windows 8**.

Die **Linux-Version** des MOPS benötigt einen Python-Interpreter in der jeweils passenden Version (gemäß Download). Mehr dazu steht in der mitgelieferten Datei mops-linux.howto.

Die jeweils aktuelle Fassung findet man hier: *www.viktorianer.de/info/info-mops.html*

Der MOPS ist **Freeware** in dem Sinne, dass die Software in der von mir angebotenen Form kostenlos verfügbar ist und nach Belieben kostenlos weitergegeben werden darf, sofern keine Änderungen vorgenommen wurden. Es handelt sich beim MOPS jedoch nicht um Open-Source-Software, d.h. der Quelltext ist nicht frei verfügbar.

1.3 Über diese Dokumentation

Diese Dokumentation ist keine Einführung in den Aufbau eines VNR oder die Abläufe innerhalb eines VNR im Allgemeinen. Ziel der Dokumentation ist lediglich die Beschreibung der Funktionsweise und Bedienung des MOPS sowie der Syntax und des Befehlsvorrats des MOPS-Assemblers. Sie richtet sich ausdrücklich an die Lehrkräfte, die den MOPS im Unterricht einsetzen, nicht an die Schüler.

2. Grundlagen der Bedienung

2.1 Die integrierte Entwicklungsumgebung (IDE)

Die Bedienung des MOPS sollte im wesentlichen intuitiv gelingen. Die Funktionen der IDE sind über das Menü, über die Taskbar (Funktionsleiste) oder über Shortcuts (Tastenkombinationen) abrufbar; die entsprechenden Shortcuts sind über die Menüeinträge ablesbar.

Der **Editor** verfügt über die üblichen Funktionen eines einfachen Quelltext-Editors. Quelltexte können geladen und gespeichert werden, in der üblichen Weise editiert und auch via „copy/cut & paste“ über die Zwischenablage kopiert, ausgeschnitten und eingefügt werden. Dazu verwendet man die üblichen Shortcuts <Strg>-<C>, <Strg>-<X> und <Strg>-<V> (bzw. auf Linux-Systemen <Strg>-<Y>).

Befindet sich ein Assembler-Quelltext im Editor, muss dieser in Maschinencode umgewandelt werden, bevor er vom VNR verarbeitet werden kann. Dies geschieht über die Funktion „**Kodieren**“. Konnte der Quelltext in Maschinencode übersetzt werden, dann ist der VNR „scharf gemacht“ – ansonsten erhält man eine detaillierte Fehlermeldung und muss den Quelltext zunächst korrigieren.

Der **Ablauf im VNR** kann nun mittels [▸]-Knopf gestartet werden. Der Maschinencode wird abgearbeitet und die dabei entstehenden Abläufe werden am Bildschirm simuliert. Der Ablauf kann jederzeit mittels [**II**]-Knopf angehalten bzw. wieder fortgesetzt oder auch mittels [■]-Taste ganz abgebrochen werden. Außerdem können automatische Haltepunkte ausgewählt und die Animationsgeschwindigkeit eingestellt bzw. die Animation ganz abgeschaltet werden – mehr dazu in Kapitel 4.2.

3. Der Assembler

3.1 Was der MOPS-Assembler (nicht) kann

Der MOPS soll vor allem eines leisten: Schülern der Sekundarstufe I möglichst anschaulich vermitteln, wie ein nach dem VNR-Prinzip gestalteter Computer im Prinzip arbeitet. Gleichzeitig sollen sie beim Arbeiten mit dem MOPS aber auch erfahren, wie sich die Programmierung einfacher Algorithmen in einer maschinennahen Assemblersprache von der Programmierung in einer Hochsprache (deren Kenntnis nicht zwangsläufig vorausgesetzt wird, aber zum Vergleich natürlich nötig ist) unterscheidet.

Nicht gedacht ist MOPS als echter Assembler mit dem entsprechenden Funktionsumfang eines echten Assemblers für aktuelle Prozessoren. Der MOPS dient eben nicht dazu, vollwertige Assemblerprogramme zu schreiben oder fortgeschrittene Assemblerprogrammierung zu erlernen. So kennt der MOPS-Assembler zwar Funktionen zur Ein- und Ausgabe, zum Lesen und Schreiben von Speicher- und Registerinhalten, arithmetische Grundfunktionen sowie bedingte und unbedingte Sprünge (und damit auch die Möglichkeit, Schleifenstrukturen aufzubauen!), aber keine indirekte Adressierung, keine Vielzahl an Registern für diverse Funktionen, keine Sammlung von Flags, keine logischen Verknüpfungen, keine Möglichkeiten der Bit-Manipulation usw. Er beschränkt sich bewusst auf das Wesentliche, um sein Ziel zu erreichen.

3.2 Die Syntax

Die Gestaltung der Syntax des MOPS-Assemblers orientiert sich grundsätzlich an typischen Elementen echter Assemblersyntax, ist aber andererseits so einfach wie möglich gehalten, um keine unnötigen Barrieren aufzubauen und die Assemblerquelltexte möglichst lesbar zu machen. Dazu gehört etwa, dass sämtliche Befehle - mit Ausnahme des abschließenden end - immer *genau einen* Operanden erwarten, dass nicht zwischen CODE- und DATA-Segment unterschieden wird, keine Definition von Variablen erforderlich ist und Sprungmarken rechts neben einem Befehl in der jeweiligen Befehlszeile stehen. Dadurch beginnen alle Befehlszeilen grundsätzlich links mit der Angabe des Befehls und es müssen keine Einrückungen beachtet werden. Konkret ist ein **syntaktisch korrekter MOPS-Assembler-Quelltext** durch folgende Eigenschaften gekennzeichnet:

- Jede ***Zeile*** enthält genau einen Befehl; dieser muss ganz links beginnen.
- Zwischen Befehl und Operand muss mindestens ein ***Leerzeichen*** stehen; ***Tabulatoren*** können als Trennzeichen ebenfalls verwendet werden.
- ***Kommentare*** werden durch Semikolon gekennzeichnet. Alles, was rechts von einem Semikolon steht, wird nicht interpretiert.
- ***Adressen*** (im VNR-Hauptspeicher) werden durch ein vorangestelltes $ (Dollarzeichen) gekennzeichnet. Der ansprechbare Adressraum für Daten ist auf die acht Adressen $64 .. $71 beschränkt. An Stelle von Adressen können auch die ***Variablen*** a .. h verwendet werden, die diesen Speicherstellen als fixe Aliase zugeordnet sind.

- ***Sprungziele*** können ***Zeilennummern*** oder selbst definierte ***Marken*** sein. Zeilennummern werden durch ein vorangestelltes # (Raute) gekennzeichnet, Marken bei ihrer Definition durch einen vorangestellten Doppelpunkt. Bei Verwendung einer Marke als Sprungziel wird nur die Marke (ohne Doppelpunkt) angegeben.
- Gültige ***Namen für Marken*** dürfen nur aus Buchstaben (ohne Umlaute und ß) und Ziffern bestehen; das erste Zeichen muss ein Buchstabe sein.
- Der gesamte Quelltext ist ***case-insensitiv***, d.h. Groß- und Kleinschreibung werden nicht unterschieden. Empfohlen wird eine durchgängige Kleinschreibung.
- Das ***Ende*** eines jeden Programms wird durch den Befehl end festgelegt.

3.3 Der Befehlssatz

Der Befehlssatz des MOPS-Assemblers umfasst insgesamt 15 Befehle. Die im Folgenden aufgeführten Befehle beschreiben den eingebauten Befehlssatz. Dieser kann vom Benutzer (gedacht ist hier an die Lehrkraft) bei Bedarf an die eigenen Vorstellungen angepasst werden: Es können für die einzelnen Befehle eigene mnemonische Codes festgelegt werden (max. 10 Zeichen, nur Buchstaben). Dazu verwendet man die optionale, mitgelieferte Datei `mops.cfg`. Wie man den MOPS mit einem eigenen Befehlssatz ausrüstet, wird in dieser Datei ebenfalls beschrieben. Der **eingebaute Befehlssatz** ist dieser:

Befehl	Code	Funktion
`ld` *adr*	10	**load**: Lade den Wert an der Adresse *adr* in den Akku
`ld` *val*	11	load: Lade den Wert *val* in den Akku
`st` *adr*	12	**store**: Speichere den Wert des Akkus an der Adresse *adr*
`in` *adr*	20	**input**: Schreibe den Wert des Eingaberegisters an die Adresse *adr*
`out` *adr*	22	**output**: Schreibe den Wert an der Adresse *adr* ins Ausgaberegister
`out` *val*	23	output: Schreibe den Wert *val* ins Ausgaberegister
`add` *adr*	30	**add**: Addiere den Wert an der Adresse *adr* zum Akku
`add` *val*	31	add: Addiere den Wert *val* zum Akku
`sub` *adr*	32	**subtract**: Subtrahiere den Wert an der Adresse *adr* vom Akku
`sub` *val*	33	subtract: Subtrahiere den Wert *val* vom Akku
`mul` *adr*	34	**multiply**: Multipliziere den Wert an der Adresse *adr* mit dem Akku
`mul` *val*	35	multiply: Multipliziere den Wert *val* mit dem Akku
`div` *adr*	36	**divide**: Dividiere den Akku durch den Wert an der Adresse *adr* ↓
`div` *val*	37	divide: Dividiere den Akku durch den Wert *val* (nur ganzzahliger Teil)
`mod` *adr*	38	**modulo**: Rest bei Division des Akkus durch den Wert an der Adresse *adr*
`mod` *val*	39	modulo: Rest bei Division des Akkus durch den Wert *val*
`cmp` *adr*	40	**compare**: Vergleiche den Akkuinhalt mit dem Wert an der Adresse *adr*
`cmp` *val*	41	compare: Vergleiche den Akkuinhalt mit dem Wert *val*
`jmp` *tar*	50	**jump**: Springe zum Zielpunkt *tar* (Zeilennummer oder Marke)
`jlt` *tar*	52	**jump if less than**: Springe ..., wenn bei cmp der Akkuinhalt kleiner war
`jeq` *tar*	54	**jump if equal**: Springe ..., wenn bei cmp der Akkuinhalt gleich war
`jgt` *tar*	56	**jump if greater than**: Springe ..., wenn bei cmp der Akkuinhalt größer war
`end`	60	**end**: Beendet ein Programm

4. Der Von-Neumann-Rechner

4.1 Technische Daten

Der VNR des MOPS hat einen Hauptspeicher mit 72 Speicherzellen, von denen per se 8 Speicherzellen für reine Datenspeicherung reserviert sind. Das entspricht zwar nicht dem reinen Von-Neumann-Prinzip, wonach Programm und Daten *ohne Unterscheidung* im gleichen (Haupt-)Speicher abgelegt werden, vereinfacht aber die Assembler-Programmierung (optionale Verwendung von Variablen statt Speicheradressen *ohne* Variablendeklaration), hilft Fehler durch Speicherkorruption zu vermeiden und trägt dazu bei, dass die Abläufe im VNR besser nachvollzogen werden können.

Weiterhin kennt der VNR nur die Zahlendarstellung im Dezimalsystem. Das ist zwar nicht sehr nah an der Realität, aber die Verwendung hexadezimalen oder gar binären Werte hätte die Zuordnung zwischen Assembler-Quelltext und VNR-Ablauf erheblich erschwert. Auch auf die vorzeichenlose Darstellung mittels Komplementärzahlen wurde aus diesem Grund verzichtet. Das hat zur Folge, dass der MOPS wegen seiner fünfstelligen Register auf den ganzzahligen (dezimalen) Zahlenbereich von -9999 bis 9999 beschränkt ist.

In der Praxis dürften die hier genannten Begrenzungen des MOPS für die vorgesehene Zielgruppe keine wirkliche Einschränkung bedeuten. Nach meiner Erfahrung aus mehreren Jahren Informatikunterricht mit dem betagten Modellrechner ALI ist die Beschränkung des MOPS auf maximal 32 (echte) Befehlszeilen Assemblerquelltext (= 64 Speicherplätze) und 8 Variablen völlig ausreichend.

4.2 Die Ablaufsteuerung

Die Grundfunktionen der Ablaufsteuerung – Start [▸], Pause [**II**], Stopp [■] – wurden bereits erläutert; komfortabler als mit den entsprechenden Knöpfen in der Funktionsleiste ist die Tastaturbedienung mittels <F5>, <Leertaste> und <Esc>.

Um den unterschiedlichen Bedürfnissen beim Einsatz des MOPS gerecht zu werden, bietet er darüber hinaus verschiedene Einstellungen für automatische Haltepunkte und die Animationsgeschwindigkeit. Die Auswahl der Animation(sgeschwindigkeit) – langsam, mittel, schnell oder aus – ist selbsterklärend. Die Modi für die Haltepunkte sind die folgenden:

- **Mikrozyklen**: Die einzelnen Abläufe setzen sich aus Mikrozyklen zusammen (wie z.B. schreibe das Ergebnis einer Rechenoperation in den Akku). Bei der Auswahl Mikrozyklus wird der Ablauf nach jedem Mikrozyklus angehalten.
- **VN-Phasen**: Der Ablauf wird nach jeder Phase eines Von-Neumann-Zyklus – Fetch, Decode, Execute – angehalten.
- **Befehlszyklen**: Der Ablauf hält nach jedem vollständig abgearbeitete Befehl (also einem vollständigen Von-Neumann-Zyklus) an.
- **Ausgabestopp**: Der Ablauf hält unmittelbar nach jedem Ausgabebefehl an.
- **Vollständig**: Das Programm läuft bis zum Ende durch.

Unabhängig vom eingestellten Modus gilt, dass beim Ausführen eines **Eingabebefehls** der Ablauf immer so lange angehalten wird, bis eine gültige Eingabe erfolgt ist.

5. Beispiel

Da diese Dokumentation für die Hand des Lehrers gedacht ist, bei dem grundlegende Kenntnisse der Zusammenhänge vorausgesetzt werden können, andererseits aber auch Schüler diese Dokumentation in die Hand bekommen können, gebe ich nur *ein* Beispiel an, das Befehle aus allen Befehlsgruppen enthält und somit zumindest der Lehrkraft ausreicht …

```
1   ; Gerade Zahlen rückwärts
2   in  a
3   ; gerade machen
4   ld  a
5   mod 2
6   cmp 0
7   ld  a
8   jeq weiter
9   sub 1
10  ; rückwärts zählen
11  st b :weiter
12  out b
13  sub 2
14  cmp 0
15  jgt weiter
16  end
```

6. Sonstiges

6.1 MOPS mit einer Assemblerdatei starten

Optional kann direkt beim Aufruf eine MOPS-Assemblerdatei als Parameter übergeben werden, deren Inhalt in den Editor übernommen wird. Nach entsprechender Verknüpfung im Dateimanager lässt sich der MOPS somit auch über (Doppel-)Klick auf eine Assembler-Datei aufrufen. Der Aufruf über die Konsole/Eingabeaufforderung sieht dann so aus:

Windows: `mops.exe beispiel.ass`

Linux: `python3 mops.pyc beispiel.ass`

6.2 Fenstergröße anpassen

Der MOPS ist so voreingestellt, dass das Programmfenster auf dem Bildschirm nach dem Starten automatisch maximiert wird. Die Größe der schematischen Darstellung des VNR wird dabei der jeweiligen Bildschirmauflösung auf dem eingesetzten PC angepasst, so dass der für den VNR vorgesehene Bereich durch die graphische Darstellung optimal ausgefüllt ist.

Zwar kann man nach dem Start die Fenstergröße in der üblichen Weise mit der Maus verkleinern, die Darstellung des VNR verändert ihre Größe dabei jedoch nicht, so dass ggf. Teile der Darstellung aus dem sichtbaren Fensterbereich verschwinden.

Möchte man mit einer anderen als der – bezogen auf die aktuelle Bildschirmauflösung – maximale Fenstergröße des MOPS arbeiten, dann kann beim Start (über die Konsole/Eingabeaufforderung) die gewünschte Fenstergröße angegeben werden, z.B. so:

Windows:
```
mops.exe -size 1024x768
mops.exe -size 1024x768 beispiel.ass
```

Linux:
```
python3 mops.pyc -size 800x600
python3 mops.pyc -size 800x600 beispiel.ass
```

Der MOPS kommt mit Auflösungen bis knapp unterhalb von 800 x 600 Bildpunkten zurecht, so dass er z.B. auch auf einem kleinen Netbook eingesetzt werden kann. Ab einer Auflösung von ca. 1280 x 1024 Bildpunkten wird der VNR nicht weiter vergrößert, weil die Darstellung dann optisch leidet. Der weiße Rand um den VNR herum wird bei höheren Auflösungen entsprechend breiter, so dass die Zeichnung immer in etwa mittig erscheint.