Die Arbeit eines Prozessors – leicht verständlich demonstriert mit *MOPS*

Diese Handreichung ist eine Essenz des offiziellen Benutzerhandbuchs für den „Modellrechner mit Pseudo-Assembler“ und hat den Anspruch, die wichtigsten Funktionen dieses Modellrechners abzudecken. Für weitere Erläuterungen wird ein Blick in das Handbuch empfohlen.

Einleitung

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“). 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.

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.

Konkret ist ein **syntaktisch korrekter MOPS-Assembler-Quelltext** durch folgende Eigenschaften gekennzeichnet:

- genau ein Befehl je Zeile
- genau ***ein Operand*** je Befehl (bis auf `end`)
- mindestens ein ***Leerzeichen*** zwischen Befehl und Operand. ***Tabulatoren*** als Trennzeichen ebenfalls erlaubt
- ***Kommentare*** beginnen mit Semikolon
- ***Adressen*** `adr` beginnen mit $ (Dollarzeichen). Adressraum für Daten von $64 bis $71. ***Variablen*** von a bis h sind auch möglich
- ***Sprungziele*** `tar` können ***Zeilennummern*** (durch # gekennzeichnet) oder selbst definierte ***Marken*** (vorangestellter Doppelpunkt) sein. Bei Verwendung einer Marke als Sprungziel wird nur die Marke (ohne Doppelpunkt) angegeben.
- ***case-insensitiver*** Quelltext, d.h. Groß- und Kleinschreibung werden nicht unterschieden. Empfohlen wird eine durchgängige Kleinschreibung.
- Das ***Ende*** eines jeden Programms wird durch den Befehl end festgelegt.

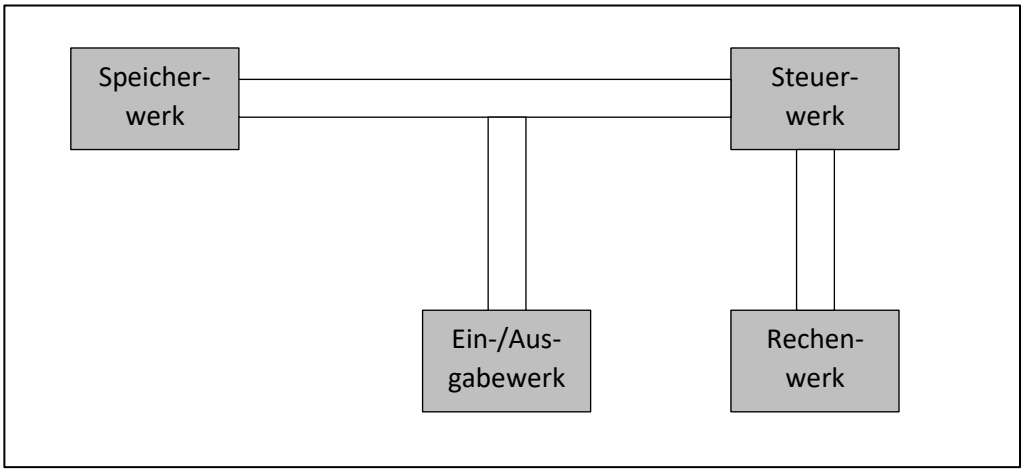
Befehlssatz

BEFEHL	CODE	FUNKTION
TRANSFERBEFEHLE		
`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 Akku 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
ARITHMETISCHE OPERATIONEN		
`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 Akku durch den Wert an der Adresse `adr`
`mod val`	39	**modulo:** Rest bei Division des Akku durch den Wert `val`
LOGISCHE OPERATIONEN		
`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`
SPRUNGBEFEHLE		
`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

Aufgaben zum Ausprobieren

1. Öffnen Sie die Datei `beispiel1.ass`. Wechseln Sie ins Vollbild, um die Statusleiste unten sehen zu können.
   a. Kodieren Sie das Programm ( ). Wählen Sie unter Ablauf „Vollständig“ und unter Animation „mittel“. Verfolgen Sie, wie der VNR arbeitet.
   Probieren Sie weitere Einstellungen für den Ablauf aus. Worin besteht der Unterschied?
   b. Analysieren Sie den Assembler-Code mithilfe der Befehlstabelle. Was kann das Programm vermutlich leisten?
   c. Vergleichen Sie den Assembler-Code mit dem entstandenen Maschinencode im Speicherwerk. Nutzen Sie die Codes in der Tabelle.
   d. Wählen Sie unter Ablauf „VN-Phasen“. Vollziehen Sie erneut einen kompletten Programmablauf nach. Beschreiben Sie auf diese Weise die Phasen, die ein von-Neumann-Rechner durchläuft.
   e. Wählen Sie unter Ablauf „Vollständig“ und unter Animation „mittel“. Lassen Sie den VNR rechnen und zählen Sie in einer Strichliste mithilfe der folgenden Abbil-

dung den Datenfluss zwischen Speicher-, Steuer-, Ein-/Ausgabe- und Rechenwerk.



Wie kann man mit den Zahlen den von-Neumann-Flaschenhals erklären?

f. Ändern Sie den Code, sodass die Division zweier eingegebener Zahlen ausgeführt und das Ergebnis ausgegeben wird.
g. Ändern Sie dieses Programm so, dass eine Division durch Null abgefangen wird (Hinweis: Sprungbefehle).

2. Öffnen Sie die Datei `betrag.ass`. Das Programm soll den Absolutbetrag einer eingegebenen Zahl zurückgeben. Führen Sie das Programm aus. Warum funktioniert es nicht so, wie es soll? Machen Sie das Programm lauffähig.

Aufgaben zur hardwarenahen Programmierung

3. Öffnen Sie die Datei `beispiel2.ass` und vollziehen Sie das Programm nach.
   Lassen Sie es vom VNR ausführen. Probieren Sie verschiedene Ablaufeinstellungen aus.

4. Schreiben Sie ein Programm, das das Minimum zweier (dreier, …) eingegebener Zahlen ausgibt.

5. Schreiben Sie einen Assemblercode für den MOPS, der die Fakultät eines eingegebenen Wertes ausgibt.

6. Gegeben ist das folgende Struktogramm zur Berechnung des größten gemeinsamen Teilers zweier Zahlen.
   Entwickeln Sie daraus ein Assemblerprogramm für den MOPS.