

Die Von-Neumann-Prinzipien erleben

Ein enaktiver Zugang zu einem abstrakten Thema

von Lisa Göbel und Lutz Hellmig

Die Von-Neumann-Prinzipien sind für viele Schülerinnen und Schüler der Sekundarstufe Bestandteil des Curriculums. Auch in den Empfehlungen zu Bildungsstandards der Gesellschaft für Informatik wird auf diese Grundlagen hingewiesen (vgl. GI, 2016, S.43 u. 45). In Sachsen (2018), in Nordrhein-Westfalen (2014), in der Schweiz (2016) oder in Mecklenburg-Vorpommern (2017) werden die Von-Neumann-Prinzipien mit Modellierungswerkzeugen veranschaulicht. In diesem Zusammenhang haben sich beispielsweise *Johnny* (vgl. Dauscher, 2013), das *Murmelrechner*-Modell (vgl. Merkert, 2012), der *Bonsai*-Lehrcomputer (vgl. Merkert, 2012) sowie der *Modellrechner mit Pseudoassembler MOPS* (vgl. Haase, 2013) in der Unterrichtspraxis bewährt.

Anlass für die Entwicklung der hier vorgestellten Unterrichtsidee zur Vermittlung der Von-Neumann-Prinzipien war der wiederholte Einsatz des MOPS (vgl. Haase, 2013). Obgleich MOPS sehr gut didaktisch reduziert und seine assemblernahe Programmierung ein-

fach strukturiert und übersichtlich ist, zeigten sich im Unterricht in der zehnten Jahrgangsstufe einer Gesamtschule trotzdem regelmäßig Verständnisprobleme bezüglich der Abläufe des Von-Neumann-Zyklus. Über eine zusätzliche enaktive Handlungsebene vor dem Einsatz des Simulationsprogramms gewinnen die Schülerinnen und Schüler einen vertieften, schülergerechten Zugang zum Aufbau und zur prinzipiellen Funktionsweise des Von-Neumann-Rechners.

Das John-von-Neumann-Modell

Der Mathematiker John von Neumann (1903–1957) hat mit seinem 1945 entwickelten allgemeinen Modell für den Aufbau und die Funktionsweise eines Computers einen nachhaltigen Einfluss in der Informatik ausgeübt (vgl. Dauscher, 2013).

Der Von-Neumann-Rechner besteht aus einem Rechenwerk, einem Steuerwerk, einem Speicherwerk und dem Eingabe-/Ausgabewerk (siehe auch Bild 1). Ein System verschiedener Busse verbindet die Komponenten miteinander und sorgt für den Datentransport (vgl. auch Claus u. a., 2006). Die Von-Neumann-Architektur sieht – im Unterschied zur Harvard-Architektur (vgl. Godse/Godse, 2011) – einen gemeinsamen Speicher für Programme und Daten vor. Ein Nachteil ist jedoch der sogenannte Von-Neumann-Flaschenhals zwischen Steuerwerk und Rechenwerk, der die Programmausführung verlangsamt (vgl. Giloi, 1993). Der Speicher enthält gleichgroße, fortlaufend nummerierte Zellen, die direkt gelesen oder beschrieben werden können.

Die Arbeitsweise eines solchen Rechners ist ebenfalls Bestandteil des Von-Neumann-Prinzips. Ein Von-Neumann-Rechner erfordert ein Programm. Dies ist eine Folge von Anweisungen, die grundsätzlich sequenziell durchlaufen werden. Es wird mindestens zwischen arithmetischen und logischen Befehlen, Transportbefehlen und unbedingten und bedingten Sprüngen unterschieden. Sprungbefehle sind die einzige Möglich-

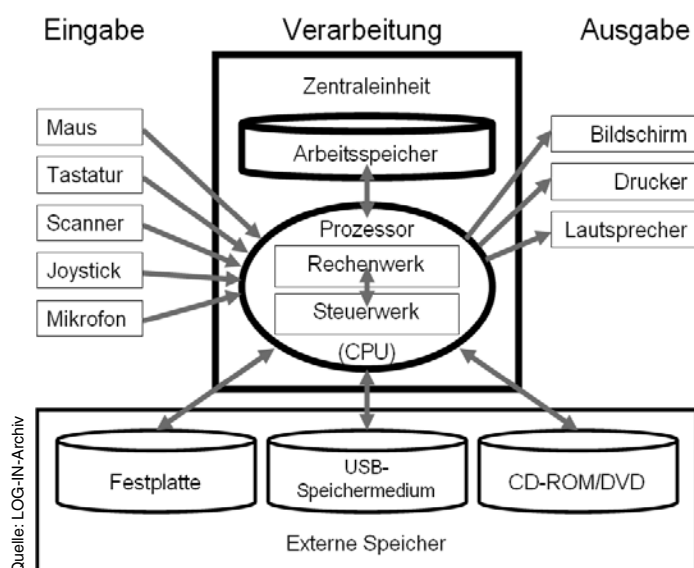
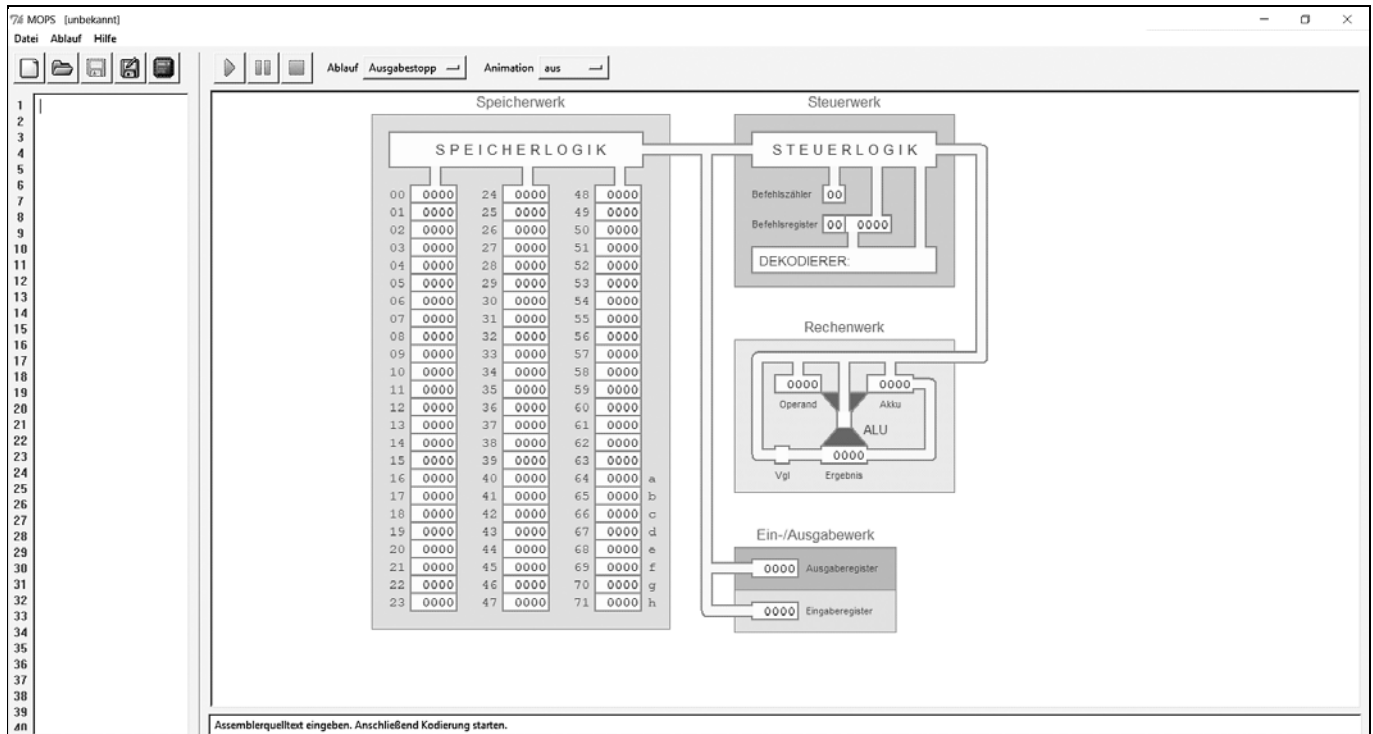


Bild 1: Grundsätzlicher Aufbau eines Rechners.



keit, die sequenzielle Abfolge der Bearbeitung zu umgehen. Schleifen oder Verzweigungen existieren nicht. Die Daten werden binär codiert.

Jeder Befehl wird in einem Befehlszyklus abgearbeitet. Durch das Starten des Programms wird der Befehlszähler, der im Steuerwerk integriert ist, auf die erste Adresse des Programms gesetzt. Anschließend läuft der Fünf-Phasen-Zyklus durch (vgl. von Neumann, 1945):

- ▷ Die erste Phase ist die *Fetch*-Phase. Hier wird ein Befehl aus dem Speicherwerk in das Befehlsregister des Steuerwerks geladen und der Befehlszähler um ein erhöht. Ein Befehl besteht jeweils aus dem Befehlscode (Operation) und ggf. einer Liste von Operanden.
- ▷ Anschließend wird in der *Decode*-Phase der Befehl im Steuerwerk dekodiert.
- ▷ In der dritten Phase – *Fetch Operands* – werden Operanden aus dem Speicher in das Datenregister geladen.
- ▷ In der *Execution*-Phase werden die Befehle im Rechenwerk ausgeführt.
- ▷ Die letzte Phase – *Write back* – dient dazu, das Ergebnis der Berechnung in Register oder Speicher zurückzuschreiben.

Dieser Zyklus wiederholt sich, bis alle Befehle aus dem Speicherwerk bearbeitet wurden. Nach Beendigung des Programms kann der Speicher gelöscht und wieder neu beschrieben werden.

Auch wenn die technische Entwicklung dazu geführt hat, dass heutige Rechner bei genauerer Betrachtung nicht vollständig und exakt den Von-Neumann-Prinzipien genügen, ist das Modell immer noch zweckmäßig, um die grundlegenden Prinzipien einer elektronischen Rechenmaschine zu verstehen.

Bild 2: Bildschirmkopie des MOPS-Programms von Marco Haase.

MOPS – ein Modellrechner mit Pseudoassembler

MOPS – ein Modellrechner mit Pseudoassembler – wurde in Anlehnung an das Prinzip des Von-Neumann-Rechners für den schulischen Einsatz in der Sekundarstufe I konstruiert, um die Vorgänge des Von-Neumann-Zyklus anschaulich simulieren zu können (vgl. Haase, 2013).

Im Bild 2 ist der Modellrechner mit den vier Komponenten des Von-Neumann-Rechners zu sehen. Bevor die Simulation ausgeführt wird, muss ein Pseudoassembler-Code in die linke Spalte des Programms geladen werden. Der Pseudoassembler nutzt einen didaktisch reduzierten Assemblercode mit insgesamt 15 Befehlen. Der Hauptspeicher besteht aus 72 Speicherzellen, wobei 64 ausschließlich für Programmcode und acht Speicherzellen exklusiv für die Datenspeicherung belegt sind. Diese Limitierung ist eine Abweichung vom Von-Neumann-Prinzip, sorgt aber für eine reduzierte Fehleranfälligkeit der Assembler-Programmierung.

Zur Erhöhung der Anschaulichkeit arbeitet der Pseudoassembler mit dem Dezimalsystem im Wertebereich von -9999 bis +9999 anstelle des Binärsystems.

Nachdem der Quellcode implementiert wurde, kann dieser in Maschinensprache übersetzt und anschließend ausgeführt werden.

In einer Leiste unterhalb des Simulators werden die einzelnen Schritte kommentiert. Hierbei werden die Begriffe des Befehlszyklus des Von-Neumann-Rechners verwendet.

Einordnung in die Stoffeinheit

Die Erprobung des Unterrichtskonzepts fand an einer Gesamtschule in der Jahrgangsstufe 10 der Einführungsphase der gymnasialen Oberstufe in Mecklenburg-Vorpommern in der Stoffeinheit „Sprachen und Sprachkonzepte verstehen“ statt.

Zu Beginn der Stoffeinheit wurden natürliche und künstliche Sprachen betrachtet sowie zwischen Sprachen und Sprachersetzungssystemen unterschieden. Die Schülerinnen und Schüler entwickelten Vorstellungen zu den Begriffen *Sprache*, *Grammatik*, *Alphabet* und *Wort*. Anhand von Beispielen erschlossen sich die Schülerinnen und Schüler Bedeutungsaspekte der Begriffe *Semantik* und *Syntax* sowie *Information* und *Daten*. Mithilfe der Simulation von Datenübertragungen in der Lernsoftware Filius (vgl. Freischlad, 2009) erkannten die Schülerinnen und Schüler die Notwendigkeit von Protokollen für den Gebrauch von Sprache. Dieser erste Themenbereich umfasste drei Doppelstunden.

Das zweite Themengebiet beinhaltete das Problemlösen am Computer mit PROLOG und JAVA. Zwei Doppelstunden lang operierten die Schülerinnen und Schüler am bewährten Beispiel des Stammbaums mit Fakten sowie einfachen und kombinierten Regeln. In zwei weiteren Doppelstunden gewannen die Schülerinnen und Schüler Erfahrungen in der textuellen imperativen Programmierung. Unter Nutzung des JAVA-Editors wurden einfache Verzweigungen sowie die while-Schleife implementiert. Für die Erschließung der elementaren Konzepte beider Programmierparadigmen wurde mit dem Prinzip Benutzen-Analysieren-Gestalten-Verankern (vgl. Hellmig/Hempel, 2017) eine Form des entdeckenden Lernens gewählt.

Aufbauend auf die Erfahrungen der Schülerinnen und Schüler aus der textuellen Programmierung wurden die besonderen Anforderungen an eine formale, für den Rechner geeignete Sprache und das Prinzip der Codierung im Dezimal-, Binär- und Hexadezimalsystems erarbeitet. Die Behandlung der Stoffeinheit schließt mit einer Doppelstunde zu den Prinzipien des Von-Neumann-Modells ab.

Unterrichtsziele

Mit der Behandlung des Von-Neumann-Modells zum Ende der Sekundarstufe I erweitern die Schülerinnen und Schüler ihre auf dem EVA(S)-Prinzip beruhenden allgemeinen Vorstellungen vom generellen Aufbau ei-

nes Rechners [EVA(S) = *Eingabe-Verarbeitung-Ausgabe (Speicherung)*].

In der Sekundarstufe I reicht es völlig aus, ein grundsätzliches Verständnis der Von-Neumann-Prinzipien bezüglich des Aufbaus und des Befehlszyklus zu vermitteln. Vertiefend kann auf den „Von-Neumann-Flaschenhals“ eingegangen werden. Mit der Untersuchung und Modifikation eines einfachen assemblernahen Programms gewinnen die Schülerinnen und Schüler erste Vorstellungen über den Charakter eines kompilierten Programms und die verschiedenen elementaren Befehlsarten. Die Entwicklung von Kompetenzen in der assemblernahen Programmierung wird nicht angestrebt. Die Modellierungswerkzeuge dienen ausschließlich zur exemplarischen Veranschaulichung eines didaktisch reduzierten Modells des Von-Neumann-Rechners.

Am Ende der Unterrichtsstunde können die Schülerinnen und Schüler den Aufbau sowie die Funktionsweise des Von-Neumann-Rechners anhand der Simulation mit MOPS erklären und wichtige Merkmale eines assemblernahen MOPS-Programms beschreiben.

Bei Wiederaufnahme des Themas „Von-Neumann-Rechner“ in der Sekundarstufe II ist eine detailliertere Betrachtung der Architektur möglich (vgl. Kerncurriculum von Berlin u. a., 2006).

Erweiterung um die enaktive Darstellungsebene

Für das abstrakte John-von-Neumann-Modell existiert eine Reihe von Visualisierungen, die den Schülerinnen und Schülern helfen, die Abläufe beim Ausführen eines Programms nachvollziehen zu können. Bei all diesen Simulationen verbleiben die Schülerinnen und Schüler in der Rolle des Betrachters einer Bildfolge oder eines kontinuierlichen Vorgangs. Nach unseren Erfahrungen ist die Verfolgung und Verinnerlichung der ablaufenden Prozessschritte trotzdem noch sehr anspruchsvoll und bei einer Reihe von Schülerinnen und Schülern nicht von Erfolg gekrönt.

Einen möglichen Lösungsansatz für das Problem bietet die Rückbesinnung auf das von Bruner (1966) formulierte Prinzip der Variation der Darstellungsebenen, das den Wechselwirkungen zwischen symbolischen, ikonischen und enaktiven Repräsentationsformen eines Gegenstands eine hohe Bedeutung für den Lernerfolg beimisst. Während symbolische und ikonische Repräsentationsformen im Unterrichtsalltag üblich sind, stellen enaktive Veranschaulichungen für gewöhnlich eine Ausnahme dar. Hartmann, Näf und Reichert (2006) klammern darüber hinaus aus der enaktiven Darstellungsebene die semi-enaktive Darstellung (die Lehrperson demonstriert enaktiv) und die virtuell-enaktive Darstellung (enaktive Vorgänge werden auf dem Computer simuliert) aus. Die verbleibende Gruppe der wirklich enaktiven, durch aktive physische Handlungen der Schülerinnen und Schüler geprägten Darstellungen

trägt im Informatikunterricht zumeist noch exotischen Charakter.

Das Voranstellen eines enaktiven Durchlaufes (im Wortsinne) verschiedener Stationen im Klassenraum nach den Regeln des Von-Neumann-Zyklus soll den Schülerinnen und Schülern erste Einsichten in den Gesamtprozess ermöglichen, die ihnen helfen, die anschließende Simulation mit MOPS besser zu verstehen.

Zum Stundenverlauf

Für die Behandlung des Themas in einer Halbgruppe einer zehnten Klasse stand jeweils eine Unterrichtseinheit von 90 Minuten Länge zur Verfügung. Neben dem Computerraum konnte auch ein normaler Klassenraum mit beweglichen Tischen und Stühlen für die enaktive Unterrichtsphase genutzt werden.

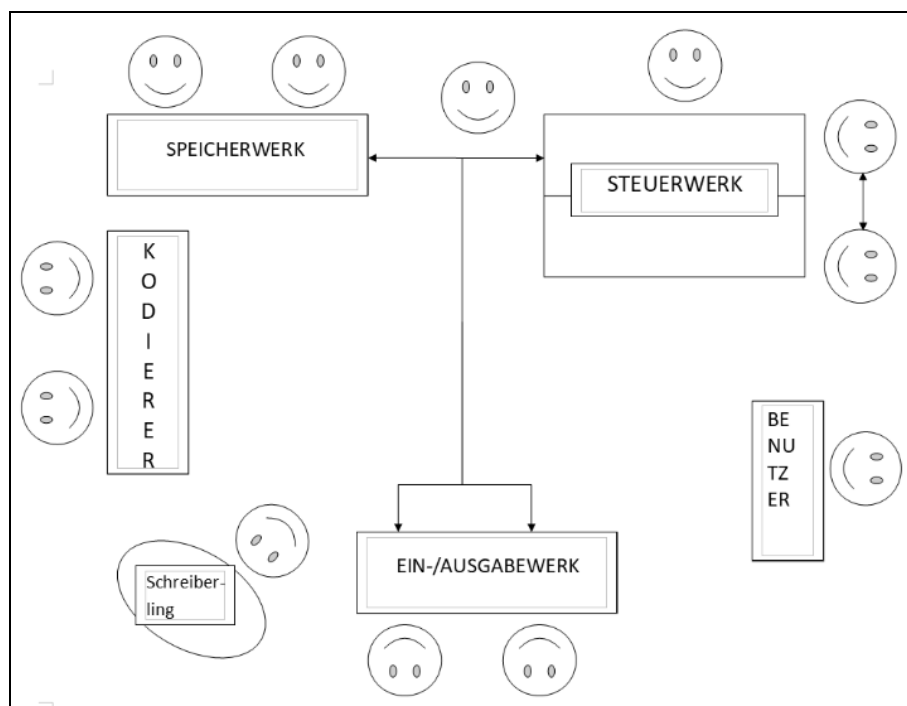
Die Stunde besteht aus vier Unterrichtsabschnitten. Nach einem kurzen Stundeneinstieg von etwa 15 Minuten wechseln die Schülerinnen und Schüler den Raum zu einem zweimaligen enaktiven Durchspielen eines minimalen Von-Neumann-Programms. In dieser 30-minütigen Phase gewinnen sie eine Vorstellung von der Komplexität der Abläufe im Inneren eines Rechners selbst bei sehr einfach scheinenden Operationen. Auf diesen Erkenntnissen aufbauend gelingt es ihnen – jetzt wieder am Rechner – in 30 Minuten ein vorgegebenes Pseudoassembler-Programm zu interpretieren und gemäß einer Aufgabenstellung zu modifizieren. Eine 10-minütige Zusammenfassung bildet das Stundende.

Stundeneinstieg

Der Stundeneinstieg beginnt mit der Zielorientierung. In den vorherigen Unterrichtseinheiten wurden Probleme mit dem Computer gelöst. Nun wird betrachtet, wie der Computer im „Inneren“ funktioniert. Nach der Motivation wird das EVA(S)-Prinzip reaktiviert. Dadurch erkennen die Schülerinnen und Schüler die Äquivalenz des Computers zum Gehirn.

Ein Bezug zur historischen Entwicklung des Von-Neumann-Rechners liefert einen Übergang vom EVA(S)-Prinzip zum Von-Neumann-Prinzip.

Bild 3: Raumaufteilung.



Enaktive Visualisierung des Von-Neumann-Modells „à la MOPS“

Mit diesem Unterrichtsabschnitt erfahren und „erlaufen“ die Schülerinnen und Schüler das Prinzip der Abarbeitung eines Programms in einem John-von-Neumann-Rechner. Durch das unmittelbare Erleben erkennen sie auch den Flaschenhals der Von-Neumann-Architektur.

Aufbau des Raums

Der gesamte Klassenraum versinnbildlicht einen John-von-Neumann-Rechner. Die im Raum platzierten Schulbänke repräsentieren die Komponenten des Von-Neumann-Rechners. Jede Schulbank ist mit ein bis zwei Schülerinnen und Schülern besetzt. Um einen späteren Transfer der Erkenntnisse aus dem Klassenraum für die Simulation mit MOPS zu unterstützen, entspricht die Anordnung der Schulbänke im Raum in etwa der Visualisierung des Von-Neumann-Rechners in MOPS.

An jeder Station liegt spezielles Material bereit, mit dem die Aufgaben der betreffenden Rechnerkomponente bearbeitet werden können.

Zusätzlich gibt es einen Schreiberling, der den Ablauf auf einem Poster dokumentiert.

Der Datentransport wird durch einen Läufer realisiert.

Vorstellung des Materials

Jede Station wird mit spezifischen Utensilien ausgestattet.

00
Code

01
Adresse

Code
Adresse

Bild 4 (links): Ausschnitt des Speicherwerks.

Bild 5 (oben): Ausschnitt des Befehlsregisters.

Bild 6 (rechts oben): Ein-/Ausgaberegister.

Ausgaberegister

Eingaberegister

Bild 7 (unten): MOPS – Modellrechner mit Pseudoassembler.

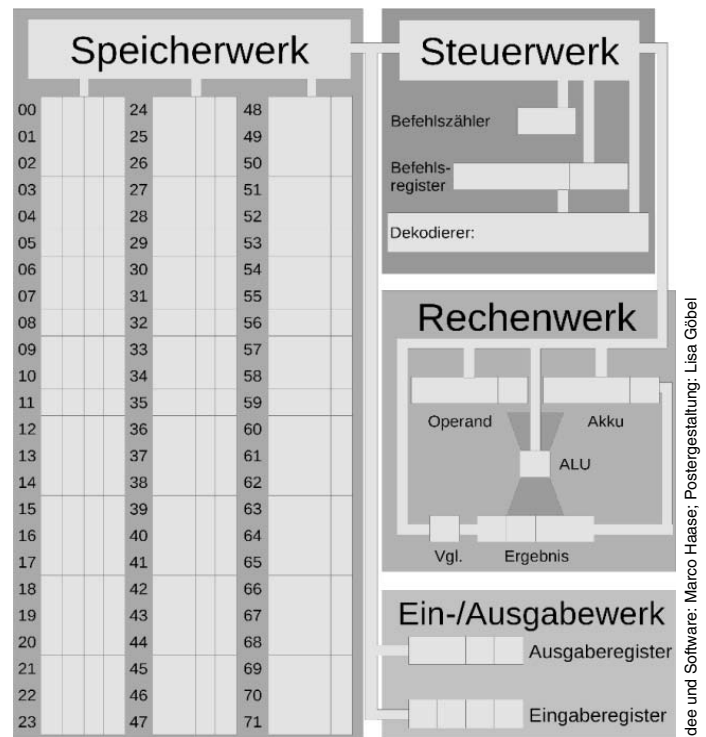
- ▷ *Speicherwerk*: Aktenordner mit Karteikarten. Im oberen Teil der Karte befinden sich jeweils die Codes, und im unteren Teil sind die Adressen. Im Bild 4 wird ein Beispiel für die erste Unterteilung gezeigt.
- ▷ *Kodierer*: siehe die folgende Funktion-Code-Tabelle.

Funktion	Code
Schreibe den Wert aus dem Eingaberegister an die Adresse <i>a</i>	20
Schreibe den Wert an der Adresse <i>a</i> in das Ausgaberegister	22
Adresse <i>a</i>	64
Ende eines Programms	60

- ▷ *Steuerwerk*: Das Steuerwerk besteht aus einem beweglichen Läufer sowie dem stationären Befehlszähler, dem Befehlsregister und dem Dekodierer. Der Befehlszähler hat ein Ringheft, bei dem auf jeder Seite eine Zahl zwischen 0 und 63 steht. Das Befehlsregister erhält einen Stift und eine Tafel mit der im Bild 5 dargestellten Form. Der Dekodierer bekommt einen Stift sowie die folgende Code-Befehlstabelle:

Code	Funktion	Befehl
20	Schreibe den Wert aus dem Eingaberegister an die Adresse <i>a</i>	in <i>a</i>
22	Schreibe den Wert an der Adresse <i>a</i> in das Ausgaberegister	out <i>a</i>
64	Adresse <i>a</i>	
60	Ende eines Programms	end

- ▷ *Benutzer*: Der Benutzer benötigt einen Stift.
- ▷ *Ein-/Ausgabewerk*: Das Ein-/Ausgabewerk erhält einen Stift sowie die im Bild 6 wiedergegebene Tafel.
- ▷ *Schreiberling*: Der Schreiberling steht neben einem Flipchart, an dem die Postergestaltung befestigt ist.



Idee und Software: Marco Haase; Postergestaltung: Lisa Göbel

Rollenvorstellung

Der Moderator bzw. die Moderatorin beschreibt zu Beginn das Ziel der Veranschaulichung. Der erste, triviale Kontext ist die Simulation eines Taschenrechners. In einem Taschenrechner soll eine Zahl eingegeben werden. Diese wird auf dem Display sofort angezeigt. Doch was passiert zwischen Eingabe und Ausgabe? Genau diese Situation soll durch diese Visualisierung simuliert werden. Dazu benötigt man einen Kodierer, ein Speicherwerk, einen Steuermann, ein Befehlsregister, einen Dekodierer, ein Ein-/Ausgabewerk, einen Benutzer und einen Schreiberling.

Das *Speicherwerk* holt Befehle und Daten aus dem Arbeitsspeicher. Befehle werden im Steuerwerk interpretiert. Daten werden im Rechenwerk verarbeitet.

Das *Steuerwerk* steuert den Ablauf der Befehlsverarbeitung. Das Steuerwerk beinhaltet einen Läufer, einen Befehlszähler sowie einen Dekodierer.

Das *Ein-/Ausgabewerk* steuert die Eingabe und Ausgabe von Daten. Das Ein-/Ausgabewerk besteht aus einem Eingaberegister und einem Ausgaberegister.

Der *Befehlszähler* zählt die Befehle, die der Läufer aus dem Speicherwerk holt. Der Befehlszähler zählt immer dann weiter, wenn der Läufer ihm dies befiehlt.

Das *Befehlsregister* übernimmt die Befehle, die der Läufer aus dem Speicherwerk holt.

Der *Dekodierer* übersetzt die Codes des Befehlsregisters in Befehle und teilt dem Läufer die Bedeutung des Befehls mit. Für das Entschlüsseln wird eine unterstützende Tabelle mit einer Auswahl von Pseudoassembler-Befehlen genutzt (siehe Tabelle 1).

Code	Funktion	Befehl
20	Schreibe den Wert aus dem Eingaberegister an die Adresse <i>a</i>	in <i>a</i>
22	Schreibe den Wert an der Adresse <i>a</i> in das Ausgaberegister	out <i>a</i>
10	Lade den Wert <i>a</i> in den Akku	ld <i>a</i>
30	Addiere den Wert an der Adresse <i>a</i> zum Akku	add <i>a</i>
12	Speichere den Wert des Akku an der Adresse <i>a</i>	st <i>a</i>
64	Adresse <i>a</i>	
60	Ende eines Programms	end

Tabelle 1: Auswahl von Pseudoassembler-Befehlen.

Der *Kodierer* gibt den kodierten Code an das Speicherwerk weiter. Bevor der Kodierer den Code weitergeben kann, muss er die Funktion kodieren. Dazu hat der Kodierer eine unterstützende Tabelle gegeben.

Der *Läufer* koordiniert die Abläufe zwischen dem Speicherwerk und dem Steuerwerk.

Der *Benutzer* gibt eine Zahl in das Eingaberegister ein.

Der *Schreiberling* notiert am MOPS-Poster jeden Schritt der Mitspieler.

Das *Rechenwerk* für die Ausführung arithmetischer und logischer Operationen wird in diesem Durchlauf noch nicht benutzt. Aufgrund der Vollständigkeit wird es trotzdem genannt.

Aufgabenstellung und Ablauf der Phase

Nach der Beschreibung der in der Halbgruppe realisierten Variante werden auch Hinweise für eine Gestaltung dieses Unterrichtsabschnitts mit einer kompletten Klasse gegeben.

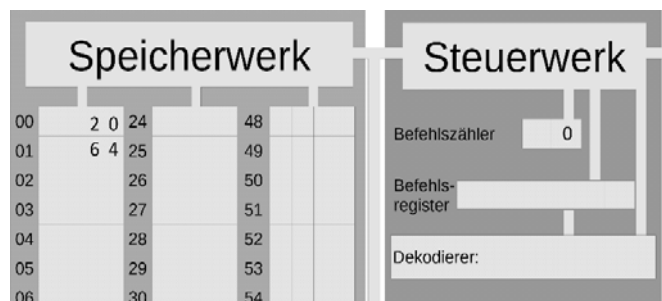
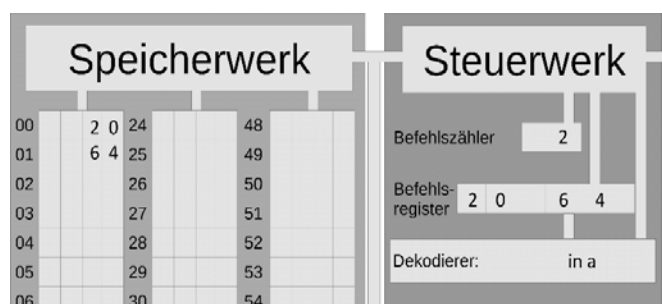


Bild 8 (oben): Ausschnitt des Posters mit Veränderung, die der Schreiberling vollzogen hat.

Bild 9 (unten): Ausschnitt des Posters des Schreiberlings.



Version 1: 15 Schülerinnen und Schüler

Die Schülerinnen und Schüler veranschaulichen enaktiv, wie eine Zahl in einem Taschenrechner eingegeben und angezeigt wird. Damit die Vorgehensweise deutlich wird, wird zunächst ein Testdurchlauf visualisiert. Im Testdurchlauf soll eine Zahl gespeichert werden.

Testdurchlauf

Der Moderator sagt dem Kodierer, dass er die Funktion „Schreibe einen Wert aus dem Eingaberegister an die Adresse *a*“ kodieren soll. Der Kodierer schaut in seiner Kodierungstabelle die Befehle nach und kodiert zu 20 und 64. Diese Codes übergibt der Kodierer genau in dieser Reihenfolge an das Speicherwerk. Das Speicherwerk übergibt die Codes in der genannten Reihenfolge an die Register (siehe auch Bild 8).

Nach dieser Vorbereitung beginnt der Haupttestdurchlauf. Der Läufer fragt den Befehlszähler, welches Register abgefragt werden soll. Der Befehlszähler beginnt bei 0 zu zählen. Er gibt dem Läufer die Rückmeldung, dass Register 0 befragt werden muss. Der Läufer geht zum Speicherwerk und möchte den Inhalt des Registers 0 haben. Das Speicherwerk zeigt den Inhalt 20 und 64. Der Läufer nimmt diesen Inhalt und gibt ihn an das Befehlsregister. Das Befehlsregister trägt den Inhalt in sein Register ein. Der Dekodierer entschlüsselt den Zahlencode 20 und 64 in den Befehlscode „in *a*“. Der Dekodierer meldet dem Läufer, dass er fertig ist und nennt den Befehlscode. Der Läufer befiehlt dem

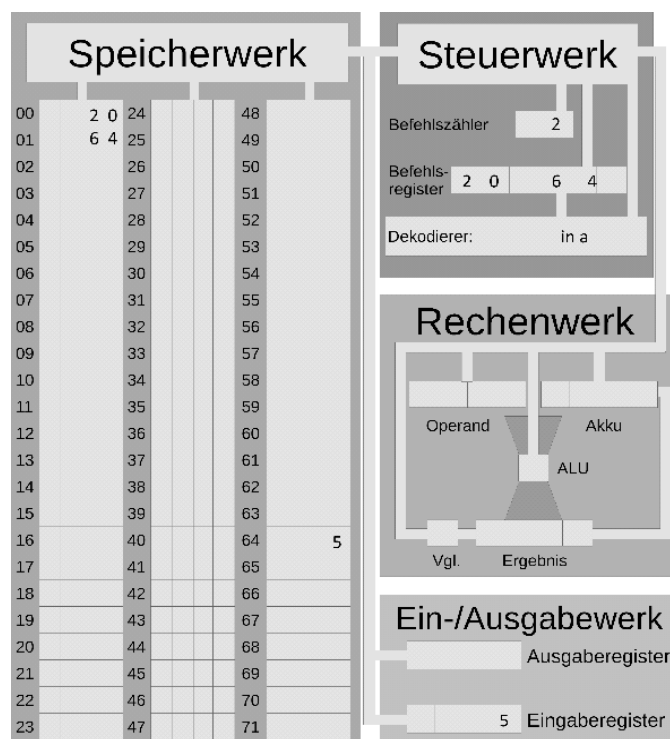


Bild 10: Poster nach Abschluss des Testdurchlaufs.

Befehlszähler weiterzuzählen. Der Befehlszähler setzt den Zähler auf 2, da bereits zwei Einträge aus dem Speicherwerk entnommen und abgearbeitet wurden (siehe auch Bild 9, vorige Seite).

Der Läufer geht nun zum Ein-/Ausgabewerk und fordert eine Eingabe vom Benutzer.

Das Eingabewerk wendet sich an den Benutzer und möchte eine Eingabe haben. Der Benutzer gibt eine beliebige Zahl ein. Das Eingabewerk gibt die eingegebene Zahl – zum Beispiel die 5 – an den Läufer weiter. Der Läufer fragt das Befehlsregister, in welchem Register die eingegebene Zahl 5 gespeichert werden soll. Das Befehlsregister meldet die Position 64. Der Läufer geht zum Speicherwerk und sagt ihm, dass diese Zahl im Register 64 gespeichert werden soll. Das Speicherwerk sucht das zutreffende Register und trägt diese Zahl ein.

Der Testdurchlauf ist abgeschlossen (siehe auch Bild 10). Das relativ simple Ziel, eine Zahl in das Speicherwerk einzutragen, wurde mit immerhin zehn Einzelschritten erreicht – eine beeindruckende Zahl für die Schülerinnen und Schüler.

Dem Testlauf schließt sich der komplette Durchlauf an. Es soll eine Zahl eingelesen und wieder ausgegeben werden. Danach endet das Programm.

Vollständiger Durchlauf

Der Moderator nennt dem Kodierer die notwendigen Funktionen: „Schreibe den Wert aus dem Eingaberegister an die Adresse a. Schreibe den Wert an der Adresse a in das Ausgaberegister. Das Programm en-

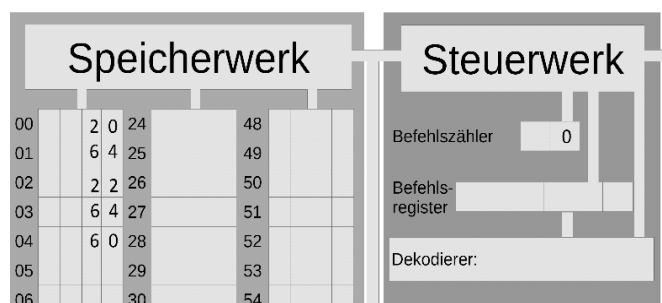


Bild 11: Poster nach den Eintragungen des Schreiberlings.

det.“ Aus dieser Funktion erstellt der Kodierer die folgenden Codes: 20, 64, 22, 64, 60.

Der Kodierer überreicht dem Speicherwerk die Codes. Dieses sortiert die Codes in die Register ein. Der Schreiberling notiert in das Poster die Codes in der Reihenfolge, wie das Speicherwerk diese einsortiert hat (siehe Bild 11).

Nach dieser Vorbereitung beginnt die konkrete enaktive Visualisierung. Es schließt sich der beschriebene Durchlauf des Testdurchlaufs an.

Nun wird dieser Vorgang der Kommunikation solange wiederholt, bis das Programm beendet wurde.

Der Läufer fragt den Befehlszähler, welches Register aus dem Speicherwerk abgefragt werden soll. Der Befehlszähler nennt das Register 2. Der Läufer geht zum Speicherwerk und möchte den Inhalt von Register 2

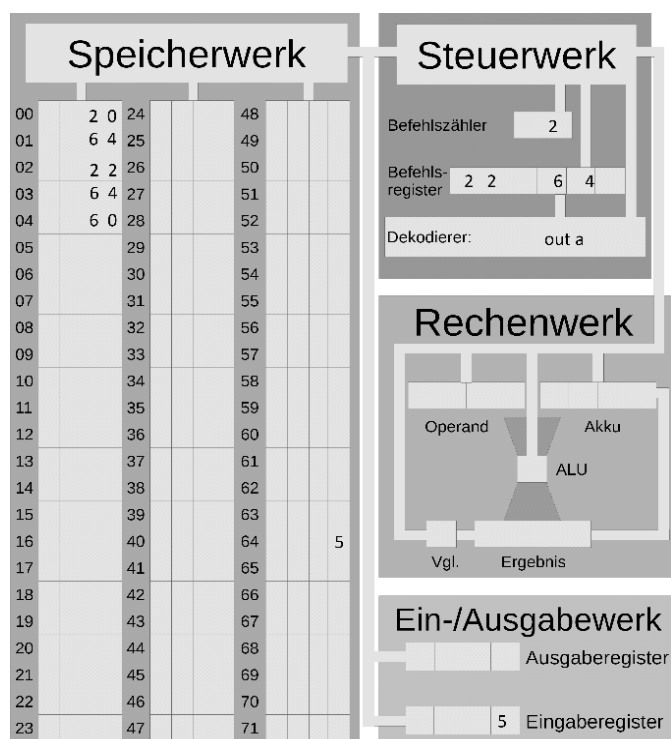


Bild 12: Poster nach Dekodierung der Codes.

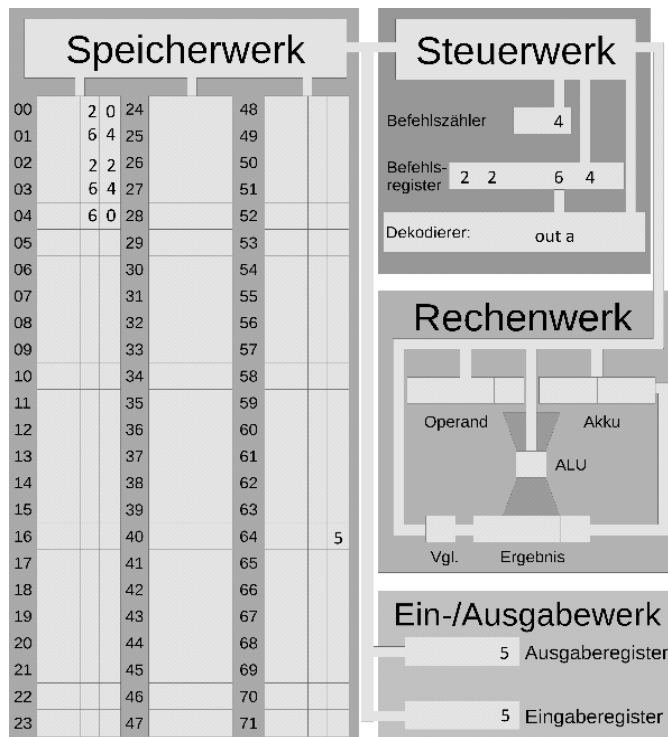


Bild 13: Poster nach der Ausgabe der Zahl.

sehen. Das Speicherwerk zeigt dem Läufer den Inhalt des Registers 2. Der Läufer zeigt dem Befehlsregister den Inhalt 22 und 64. Das Befehlsregister trägt den Inhalt in sein Register ein. Es befiehlt dem Dekodierer, die Codes zu dekodieren. Der Dekodierer dekodiert den Code 22, 64 zu „out a“. Er gibt dem Läufer den Befehl und meldet ihm dadurch, dass er seine Arbeit erledigt hat (siehe auch Bild 12).

Der Läufer sagt dem Befehlszähler, dass er weiterzählen soll. Der Befehlszähler setzt die Zähler auf 4, da wieder zwei Register aus dem Speicherwerk bearbeitet wurden.

Der Läufer fragt das Befehlsregister, in welchem Register sich die auszugebende Zahl befindet. Das Befehlsregister zeigt die Registernummer 64. Der Läufer wendet sich an das Speicherwerk und möchte den Inhalt des Registers 64 sehen. Das Speicherwerk sucht das Register 64 und zeigt dem Läufer den Inhalt 5. Der Läufer geht zum Ein-/Ausgabewerk und zeigt dem Ausgaberegister die Zahl 5. Das Ein-/Ausgabewerk trägt diese Zahl in das Ausgaberegister ein. Dem Benutzer wird die Zahl gezeigt (siehe Bild 13).

Der Läufer fragt den Befehlszähler, welches Register abgefragt werden soll. Der Befehlszähler nennt die Zahl 4. Der Läufer geht zum Speicherwerk und möchte den Inhalt von Register 4 sehen. Das Speicherwerk sucht das Register und zeigt den Inhalt 60. Der Läufer sagt dem Befehlsregister den Inhalt 60. Das Befehlsregister trägt den Code 60 in sein Register ein. Er befiehlt dem Dekodierer diesen Code zu entschlüsseln. Der Dekodierer dekodiert den Code zu „end“. Der Dekodierer sagt dem Läufer, dass dieser Code das

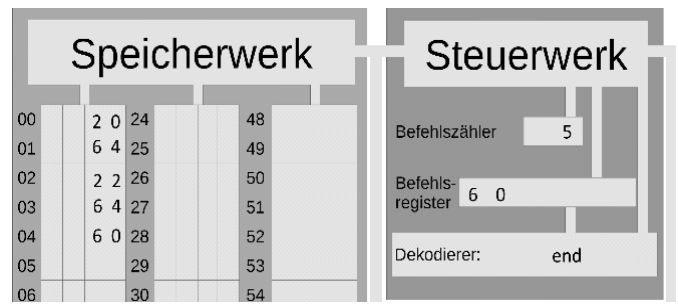


Bild 14: Endzustand des Posters.

Ende des Programms bedeutet. Der Läufer sagt dem Befehlszähler, dass er weiterzählen soll. Dieser setzt den Zähler nun auf 5, da dieses Mal nur ein Code aus dem Register entnommen wurde. Der Durchlauf ist damit beendet (siehe Bild 14).

Version 2: 30 Schülerinnen und Schüler

Da nicht jede Schule den Informatikunterricht in Halbgruppen anbieten kann, werden einige Anregungen gegeben, um die enaktive Visualisierung im gesamten Klassenverband zu veranschaulichen.

Eine Möglichkeit ist, die Klasse aufzuteilen. Die Visualisierung wird in zwei verschiedenen Räumen gleichzeitig durchgespielt. Der Ablauf bleibt jeweils gleich.

Bei einer nicht ganz so großen Klassenstärke ist es möglich, bestimmte Komponenten doppelt zu belegen und so Teamarbeit zu fördern.

Auswertung der enaktiven Visualisierung

Nach Beendigung reflektieren die Schülerinnen und Schüler den Durchlauf und stoßen auf den Flaschenhals des Von-Neumann-Modells.

Der Begriff *Von-Neumann-Flaschenhals* (engl.: *Von Neumann bottleneck*), wurde von John W. Backus (1924–2007) geprägt, der ihn 1977 in einem Vortrag einführte (Backus, 1978, S.615):

Surely there must be a less primitive way of making big changes in the store than by pushing vast numbers of words back and forth through the von Neumann bottleneck. Not only is this tube a literal bottleneck for the data traffic of a problem, but, more importantly, it is an intellectual bottleneck that has kept us tied to word-at-a-time thinking instead of encouraging us to think in terms of the larger conceptual units of the task at hand. Thus programming is basically planning and detailing the enormous traffic of words through the von Neumann bottleneck, and much of that traffic concerns not significant data itself, but where to find it.

[Deutsch: Sicherlich muss es auf eine weniger primitive Art möglich sein, große Änderungen auf dem Speicher durchzuführen, als riesige Mengen von Datenwörtern vor und zurück durch den Von-Neumann-Flaschenhals zu schieben. Diese Röhre bildet nicht nur einen wörtlichen Flaschenhals für den Datenverkehr eines Problems, sondern, was noch

wichtiger ist, es ist ein intellektueller Flaschenhals, der uns an ein Denken „ein Datenwort auf einmal“ gebunden hat, anstatt uns zu ermutigen, in den Begriffen der größeren konzeptuellen Einheiten der vorliegenden Aufgabe zu denken. Folglich ist Programmieren im Kern das Planen und Ausarbeiten des enormen Verkehrs an Datenworten durch den Von-Neumann-Flaschenhals, und ein großer Teil dieses

Verkehrs betrifft nicht die signifikanten Daten selbst, sondern wo diese zu finden sind.]

Die Veranschaulichung ist erweiterbar. Es ist möglich, das Rechenwerk einzubinden. Jedoch wird die Durchführung dadurch komplexer und zeitaufwendiger.

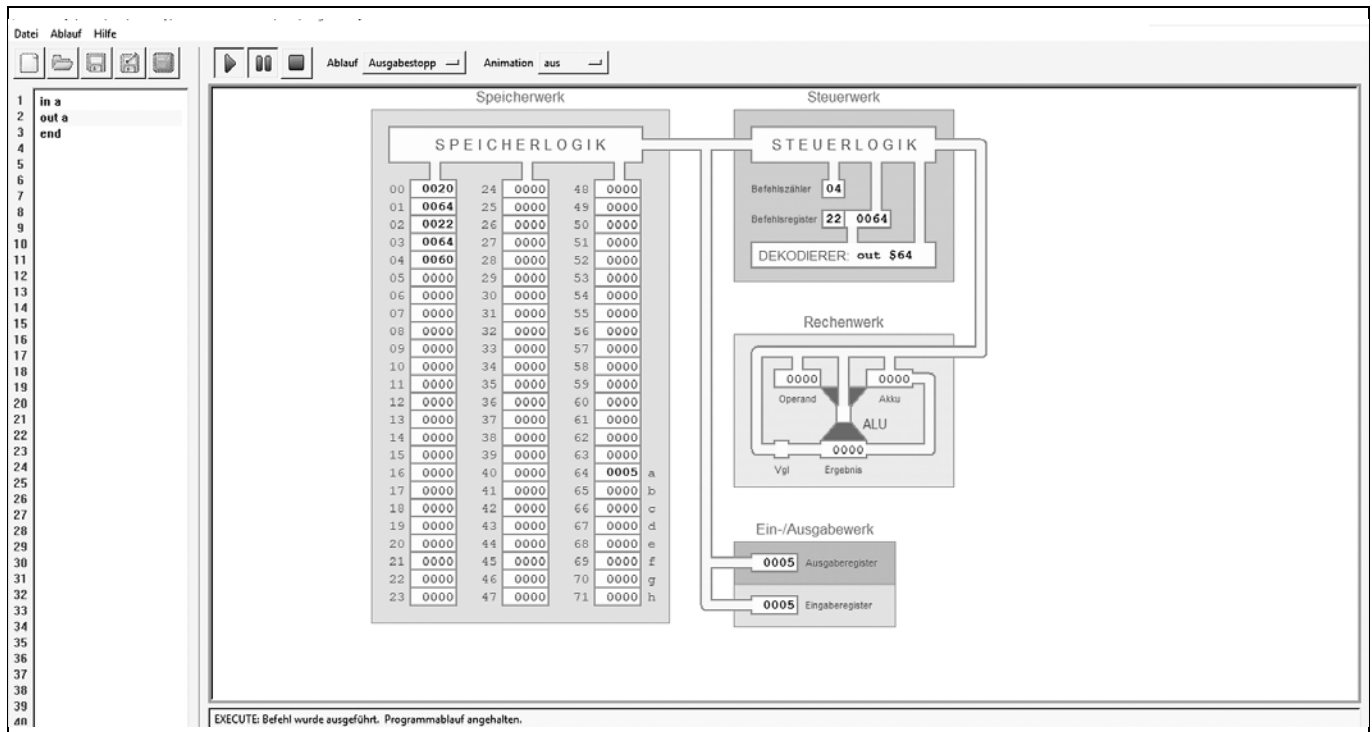
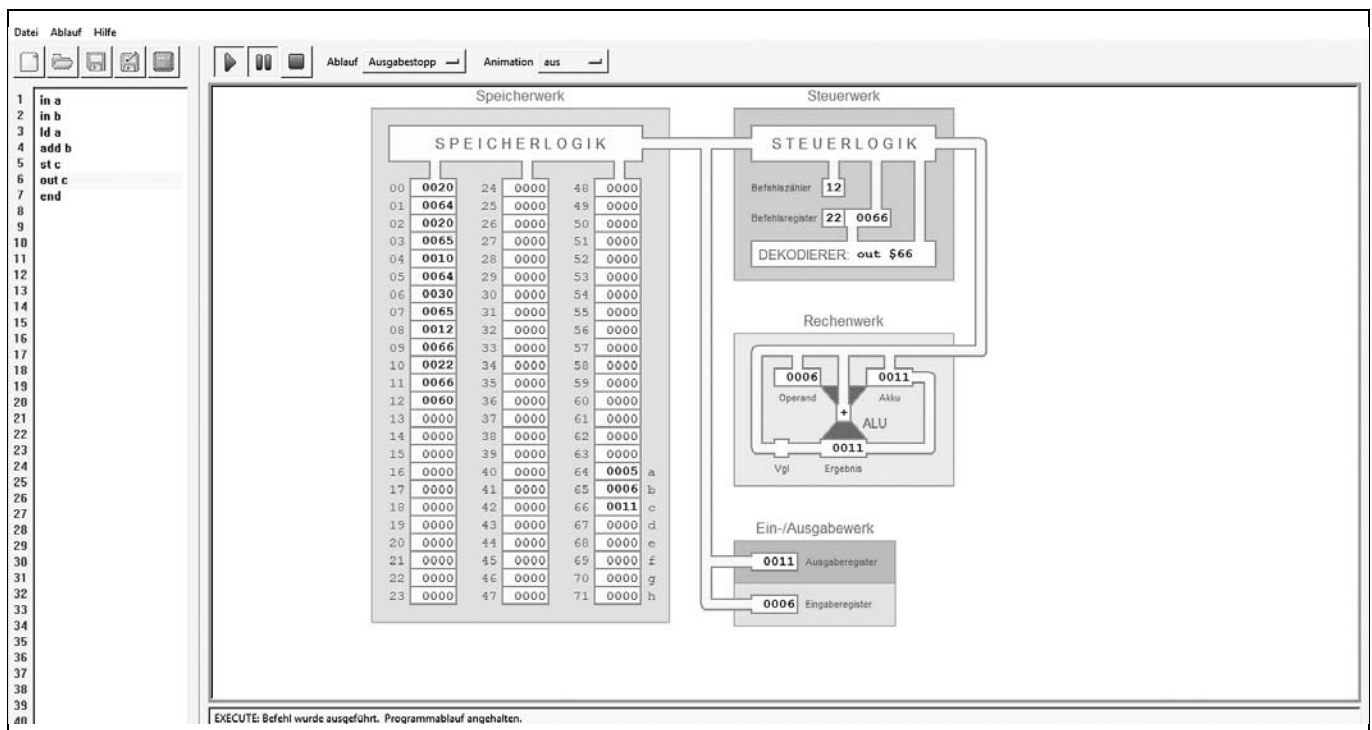


Bild 15 (oben):
Bildschirmkopie des Eingabe-Ausgabe-Programms mit MOPS.

Bild 16 (unten):
Bildschirmkopie der Addition von zwei Zahlen mit MOPS.



Assemblerprogrammierung mit MOPS am Computer

Durch die enaktive Visualisierung fällt es den Schülerinnen und Schülern nun leichter, das Programm MOPS zu nutzen. Zu beachten ist, dass jede Schülerin und jeder Schüler eine Befehls-Code-Tabelle erhält.

Am Computer öffnen die Schülerinnen und Schüler zuerst das Programm eingabe.ass und lassen es durchlaufen.

Die Schülerinnen und Schüler beschreiben ihre Beobachtungen und stellen eine Verbindung zur enaktiven Visualisierung aus der vorherigen Phase her.

Im Folgenden bekommen die Schülerinnen und Schüler die Aufgabe, das Programm durch Modifikation so zu verändern, dass zwei beliebige einzugebende Zahlen addiert werden können (siehe auch Bild 16, vorige Seite).

Die Schülerinnen und Schüler modifizieren das Programm selbstständig mithilfe der Befehls-Code-Tabelle. Sie testen nach der enaktiven Visualisierung eifrig das Programm und finden ohne fremde Hilfe die Funktionsweise des Programms heraus.

Zur Sicherung von Zwischenergebnissen stellen einzelne Schülerinnen und Schüler ihre Lösungen und Vorgehensweisen vor.

Fazit

Nach der Durchführung der Unterrichtsstunde wurden die Schülerinnen und Schüler nach einem Feedback zu der enaktiven Visualisierung befragt. Die Schülerinnen und Schüler empfanden die Unterrichtsstunde als sehr abwechslungsreich. Die Arbeit ohne Computer trug zu einem besseren Verständnis der Vorgänge durch die Schülerinnen und Schüler bei. Einige Schülerinnen und Schüler hätten sich einen weiteren Durchlauf der enaktiven Visualisierung gewünscht. Die Schülerinnen und Schüler betonten, dass sie es als schwierig empfinden würden, wenn sie ohne die enaktive Visualisierung die Assemblerprogrammierung mit MOPS durchführen müssten. Das Verständnis für die Assemblerprogrammierung hat sich durch die Enaktivität geöffnet.

Die enaktive Visualisierung des Von-Neumann-Rechners „à la MOPS“ ist für die Schülerinnen und Schüler eine große Abwechslung. Sie sind sehr motiviert, eine neue bzw. andersartige Unterrichtsform zu erfahren. Neben der Variation der Unterrichtsform ist der Lerngehalt der Unterrichtsstunde gegeben. Das eigene Handeln fördert die Abstraktion des neuen Sachverhalts und die Nachhaltigkeit des Wissenserwerbs. Die Schülerinnen und Schüler erkennen den Flaschenhals des Von-Neumann-Rechners, da sie diesen am eigenen Handeln erfahren haben.

Lisa Göbel
Dr. Lutz Hellmig

Didaktik der Informatik
Lehrstuhl für Praktische Informatik
Universität Rostock
18051 Rostock

E-Mail: lisa.goebel@uni-rostock.de
E-Mail: lutz.hellmig@uni-rostock.de

LOG-IN-Service: Die zu diesem Beitrag gehörenden Materialien und MOPS-Dateien stehen über den LOG-IN-Service zum Herunterladen zur Verfügung (siehe Seite ### in diesem Heft).

Literatur und Internetquellen

Backus, J.: Can Programming Be Liberated from the von Neumann Style? A Functional Style and Its Algebra of Programs. In: Communications of the ACM, Band 21 (1978), Nr. 8, S.615–641.
<https://t1p.de/avp8>

Bruner, J.S.: Toward a Theory of Instruction. Cambridge (MA, US): Harvard University Press, 1966 [deutsch: Entwurf einer Unterrichtstheorie. Berlin; Düsseldorf: Berlin Verlag; Pädagogischer Verlag Schwann, 1974].

Bruner J.S.; Olver, R. R.; Greenfield, P.M.: Studies in Cognitive Growth. New York (NY, USA): Wiley, 1966 [deutsch: Studien zur kognitiven Entwicklung. Stuttgart: Klett, 1971].

Claus, V. (Bearb.); Schwill, A. (Bearb.); Böving, R. (Red.): Duden Informatik A–Z – Fachlexikon für Studium, Ausbildung und Beruf. Mannheim u. a.: Dudenverlag, 2006, Stichwort: Von-Neumann-Rechner.

Dauscher, P.: Aufbau und Funktionsweise eines Von-Neumann-Rechners – Ein möglicher Unterrichtsgang mit dem Open-Source-Simulator Johnny. In: LOG IN, 33. Jg. (2013), Nr. 175, S.54–61.

Freischlad, St.: Filius. 2009 ff.
<http://www.lernsoftware-filius.de/Startseite>

Giloi, W.K.: Rechnerarchitektur. Reihe „Springer Lehrbuch“. Berlin; Heidelberg: Springer, 2009.

Godse, A.P.; Godse, D.A.: Computer Organization & Architecture. Pune (IN-MH): Technical Publications, 2011.

Haase, M.: Modellrechner MOPS. 2013 ff.
<http://www.viktorianer.de/info/mops.html>

Hartmann, W.; Näf, M.; Reichert, R.: Informatikunterricht planen und durchführen. Reihe „eXamen.press“. Berlin; Heidelberg: Springer, 2006.

Hellmig, L.; Hempel, T.: Benutzen – Analysieren – Gestalten – Verankern. Ein didaktischer Vierschritt im Informatikunterricht. In: LOG IN, 37. Jg. (2017), Nr. 187/188, S.89–96.

Merkert, J.: Funktionsweise eines Rechners. 2012 ff.
<http://inf-schule.de/rechner>

von Neumann, J.: First Draft of a Report on the EDVAC. Philadelphia (PA, USA): University of Pennsylvania: 1945. Nachdruck in: IEEE Annals of the History of Computing, Band 15 (1993), Heft 4, S.27–75.
<https://t1p.de/j58b>

Curriculare Grundlagen

Berlin u. a.:
Kerncurriculum für die Qualifikationsphase der gymnasialen Oberstufe – Informatik. Berlin, Land Brandenburg, Mecklenburg-Vorpommern: 2006.

Gesellschaft für Informatik:

GI – Gesellschaft für Informatik (Hrsg.): Bildungsstandards Informatik für die Sekundarstufe II. Erarbeitet vom Arbeitskreis „Bildungsstandards SII“ unter Koordinierung von Gerhard Röhner – Empfehlungen der Gesellschaft für Informatik e. V. vom 29.01.2016. In: LOG IN, 36. Jg. (2016), Nr. 183/184, Beilage.
<https://t1p.de/2myc>

Mecklenburg-Vorpommern:

Curriculares Konzeptpapier für die Einführung des Fachs Informatik und Medienkunde in den Jahrgangsstufen 5 bis 10 an Regionalen Schulen, Gesamtschulen und Gymnasien im Modellvorhaben „Integrierte Berufsbildung“. Schwerin: Ministerium für Bildung, Wissenschaft und Kultur Mecklenburg-Vorpommern, Juli 2017.

Nordrhein-Westfalen:

Kernlehrplan für die Sekundarstufe II Gymnasium/Gesamtschule in Nordrhein-Westfalen. Düsseldorf: Ministerium für Schule und Weiterbildung des Landes Nordrhein-Westfalen, 2014.
<https://t1p.de/mpe0>

Sachsen:

Lehrplan Gymnasium Informatik. Dresden: Staatsministerium für Kultus und Sport des Freistaats Sachsen, 2004/2007/2011/2018.
<https://t1p.de/kyhz>

Schweiz:

Arbeitsgruppe Grundlagenfach Informatik Schweiz: Lehrplanentwurf Grundlagenfach Informatik. Version vom 7. Juli 2016.
<https://t1p.de/yapj>

Alle Internetquellen wurden zuletzt am #■#. #■# 2019 geprüft und können auch aus dem Service-Bereich des LOG IN Verlags (<http://www.login-verlag.de/>) heruntergeladen werden.

Anzeige