Schriftliche Hausarbeit zur Abschlussprüfung der erweiternden Studien für Lehrer im Fach Informatik

VIII. Weiterbildungskurs in Zusammenarbeit mit der Fernuniversität Hagen

Eingereicht dem Amt für Lehrerausbildung -Außenstelle Gießen -

Vorsitzender des Prüfungsausschusses: Jungermann

Die Programmierung einer Registermaschinensimulation unter objektorientierten Gesichtspunkten

Verfasser: Tilo Böttcher

Gutachter: Dr. Jürgen Poloczek

Inhaltsverzeichnis

1. Einleitung

Bei der Planung und Themenfindung für die Hausarbeit bin ich von folgender Überlegung ausgegangen:

Wie kann man Informatikunterricht in der heutigen Zeit erfolgreich umsetzen?

Allgemeine didaktische Überlegungen gehen davon aus, dass dem Schüler im Lernprozess eine aktive Rolle zukommt. In der Didaktik der Informatik müssen sich diese Ansätze natürlich wiederfinden. Schubert und Schwill beschreiben erfolgreichen Informatikunterricht als Unterricht der einen Lernprozess anstrebt, *„der vom aktiven Schüler ausgeht, der sich Informatik handlungsorientiert, problemorientiert, anwendungsorientiert und ganzheitlich aneignet und dabei vom Lehrer begleitet wird.“*[1]

Neben den didaktischen Vorgaben und den fachlichen Inhalten des Lehrplans muss das Unterrichtsthema auch den im Lehrplan angegebenen allgemeinen Zielen des Informatikunterrichtes genügen. Dort werden unter anderem genannt:

- Analyse, Beschreibungen und Modellierung komplexer Systeme
- Problemlösungsmethoden und ihre Bewertung
- Schöpferisches Denken und Motivation
- Kommunikative und kooperative Arbeitsformen [2]

Unter diesen Voraussetzungen scheint mir das Thema *„Programmierung einer Registermaschinensimulation unter objektorientierten Gesichtspunkten“* als sehr passend.

Man kann so den Schüler und Schülerinnen eine interessante und motivierende Herausforderung bieten, den didaktischen Überlegungen gerecht werden und gleichzeitig Inhalte der objektorientierten Modellierung aus 12/I und den Konzepten und Anwendungen der Theoretischen Informatik aus 13/I miteinander verbinden.

Dies gelingt aber nur, wenn der Unterricht mit geeigneten Methoden durchgeführt wird, die den aktiven Schüler in das Zentrum des Unterrichts stellten. *„Unterschätzt wird meist, dass mehr Schüleraktivität einen deutlich höheren Vorbereitungsaufwand des Lehrers voraussetzt. Was so leicht aussieht bei Unterrichtsbeobachtungen , die den Lehrer als Impulsgeber im Hintergrund zeigen, funktioniert nur bei starker konzeptioneller Vorleistung.“*[3]

Diese Hausarbeit soll eine Anregung zur Vorbereitung und Gestaltung von Informatikunterricht sein, um allgemeine didaktische und fachspezifische Anforderungen umzusetzen. Mein Ziel ist es diese Überlegungen zwar exemplarisch an einem Thema darzustellen, sie aber so zu formulieren, dass sie ohne großen Aufwand auf andere Themen methodisch und didaktisch übertragbar sind.

[1] [Schubert / Schwill] S.33
[2] [Lehrplan] S.3
[3] [Schubert / Schwill] S.33

2. Einordnung in den Lehrplan und unterrichtliche Vorraussetzungen

Das Thema „Programmierung einer Registermaschinensimulation unter objektorientierten Gesichtspunkten“ kann unter verschiedenen Voraussetzungen behandelt werden.

Wie die Bezeichnung des Themas vermuten lässt, sollten die Schüler und Schülerinnen über Kenntnisse in der objektorientierten Programmierung verfügen und die Funktionsweise einer Registermaschine kennen.
Dies ist aber nicht zwingend erforderlich. Man kann je nach Intension an diesem Beispiel den Schwerpunkt auf die objektorientierte Programmierung oder die Funktionsweise der Registermaschine und ihre Bedeutung für die theoretische Informatik legen.
Je nach Einordnung des Themas in den Lehrplan geht man von unterschiedlichen Voraussetzungen aus. Ich möchte an dieser Stelle auf verschiedene Einsatzmöglichkeiten in der Sekundarstufe II eingehen.
Ich werde in dieser Arbeit den Schwerpunkt auf die objektorientierte Programmierung legen und die Variante 2.1. später ausführlich vorstellen.

2.1. Einordnung in den Lehrplan 12/I

Das Thema objektorientierte Programmierung ist Inhalt der Jahrgangsstufe 12/1 in Hessen. In diesem Zusammenhang liegt der Schwerpunkt meines Themas in der objektorientierten Analyse und dem objektorientierten Design von Softwareprojekten. Die Registermaschine ist als Softwareprojekt weniger für den Einstiegsunterricht als eher zur Festigung und Übung geeignet.
Möchte man diesen Weg beschreiten, würde sich das Thema im Lehrplan 12/I in den Grundkonzepten des Software-Engineerings einordnen.

In diesem Fall müssen die Schüler und Schülerinnen die grundlegenden Programmierkenntnisse aus 11/II beherrschen und diese in einer Programmiersprache umsetzen können. Des weiteren sollten sie aus 12/I das Objektmodell und den Begriff der Klassen mit ihren Attributen, Methoden und Beziehungen kennen. Von Vorteil sind auch Kenntnisse in der graphischen Modellierungssprache UML.

Da in 12/I die theoretische Informatik noch nicht behandelt wurde, ist es notwendig, den Aufbau und die Funktionsweise der Registermaschine den Schülern und Schülerinnen bekannt zu machen. Am geeignetsten erscheint mir, dass die Schüler und Schülerinnen sich dieses Wissen im Rahmen der Programmieraufgabe selbstständig mit geeigneten Materialien erarbeiten. Dabei sollte der Aufbau und die Funktionsweise der Registermaschine als Simulation einer theoretischen, programmierfähigen Maschine im Vordergrund stehen. Darauf werde ich an späterer Stelle genauer eingehen. Die Bedeutung der Registermaschine für die theoretische Informatik kann an dieser Stelle vernachlässigt werden, da dies in 13/I ausführlich behandelt wird. An dieser Stelle zeigt sich, dass die Wahl der Registermaschine als Programmierthema in 12/I den Vorteil hat, dass die Schüler und Schülerinnen in 13/I die Registermaschine aus einer anderen Sichtweise und auf einem anderen Niveau behandeln und so Inhalte des Informatikunterrichtes kursübergreifend deutlich werden.
Ungeeignet erscheinen mir Lehrervortrag oder Schülerreferate zur Vermittlung des notwendigen Wissens zur Registermaschine, da dies in 12/I für die Schüler und Schülerinnen

„von Oben“ verordnet erscheinen wird und dem Ziel der aktiven Schüler und Schülerinnen widerspricht.

2.2. Einordnung in den Lehrplan 13/I

In 13/I sieht der Lehrplan in Hessen „Konzepte und Anwendungen der theoretischen Informatik“ vor. In dieses Thema ordnet sich die Registermaschine und die Turingmaschine ein. Hierbei geht es um

- Turing- oder Registerberechenbar
- Churchsche These
- Computer als universelle symbolverarbeitende Maschine[4]

Die Programmierung einer Registermaschine bietet sich in diesem Thema nur um eine Simulationsumgebung zu schaffen an der man die verschiedenen Aspekte der theoretischen Informatik untersuchen kann.
Es würde sich in diesem Zusammenhang anbieten, dass den Schülern und Schülerinnen die Turingmaschine bekannt ist und sie einige Erfahrungen im Umgang mit einer Turingmaschinensimulation besitzen. Darauf aufbauend könnten die Schüler selbstständig ihr Wissen über die Turingmaschine auf die Registermaschine übertragen. Der Schwerpunkt der Aufgabenstellung sollte in diesem Fall ein Problem der theoretischen Informatik sein, bei dem die Programmierung einer Simulationsumgebung notwendig ist und der Computer als Medium genutzt wird.

Ein Vorteil des Einsatzes in 13 /I wäre, dass die Schüler und Schülerinnen ihre Kenntnisse aus 11/II und 12/I anwenden, mit den Erfordernissen aus 13/I verknüpfen und somit erkennen, dass Programmierung nicht zum Selbstzweck erlernt wird. Dies kann in einem abschließenden Projekt oder in Gruppenarbeit erfolgen.

Ein überwiegender Nachteil wäre aber aus meiner Sicht, dass die Probleme der programmtechnischen Umsetzung zeitlich und inhaltlich die eigentlichen Ideen der theoretischen Informatik überlagern würden.

Diesen Ansatz werde ich aus diesem Grund in dieser Arbeit nicht verfolgen und da der Schwerpunkt auf der objektorientierten Programmierung und nicht auf der theoretischen Informatik liegt.

2.3. Weitere Einsatzmöglichkeiten

Das Thema Programmierung einer Registermaschine lässt sich auch noch an anderer Stelle einordnen. In 13/II gibt es das Wahlthema „Simulationen“. Im Zusammenhang mit der Modellbildung ist die Programmierung der Registermaschine eine Möglichkeit an Hand des kursübergreifenden Themas bekannte Inhalte aus 12/I und 13/I zu verbinden und eine geeignete, für die Schüler nachzuvollziehende Aufgabenstellung.
Des weiteren kann das Thema unter Variierung der Aufgabenstellung in Projektwochen, als besondere Lernleistung (unter Beachtung der theoretischen Informatik) oder als Präsentationsthema im 5. Prüfungsfach genutzt werden.

[4] [Lehrplan]

3. Fachwissenschaftliche Überlegungen

3.1. Die Registermaschine als Computermodell

Die Registermaschine ist ein einfaches Rechnermodell, das dem Konzept eines universellen Rechenautomaten entspricht. Dieses Konzept wurde bereits 1830 von Charles Babbage mit seiner *Analytic Engine* entwickelt, in der *„zum ersten Mal die Idee des vom materiellen Gerät unabhängigen Programms“*[5] dargestellt wird. Alan M. Turing entwickelte diese Idee der Universalität weiter. Er schreibt: *„Die Existenz von Maschinen mit der Eigenschaft der Universalität hat die wichtige Konsequenz, dass es ... unnötig ist immer neue Maschinen für unterschiedliche Rechenprozesse zu entwickeln. Diese können allesamt mit einem einzigen Digitalrechner durchgeführt werden, der für jeden Fall geeignet zu programmieren ist. Es wird sich zeigen, dass infolge dessen alle Digitalcomputer in gewisser Hinsicht äquivalent sind.“*[6]

Die Registermaschine wird wie folgt definiert:
Eine Registermaschine basiert auf der idealisierten Annahme eines unbeschränkten **Speichers**, der sich aus unendlich vielen **Registerzellen** zusammensetzt, die jeweils eine natürliche Zahl speichern können. Die Registerzellen sind mit 1 beginnend durchnummeriert. Der Inhalt der Registerzelle wird mit c(i) bezeichnet.
Die Registermaschine verfügt über einen **Akkumulator**, in dem sämtliche Operationen ausgeführt werden.
Die Arbeitsweise einer Registermaschine wird durch ein **Programm** spezifiziert. Dieses setzt sich aus einer Folge von elementaren Befehlen zusammen, die Zeilenweise untereinander geschrieben werden. Die Programmzeilen sind mit Marken (Zeilennummern) versehen.
Ein Befehlszähler gibt Aufschluss darüber, welcher Befehl als nächstes zur Bearbeitung ansteht. [7]

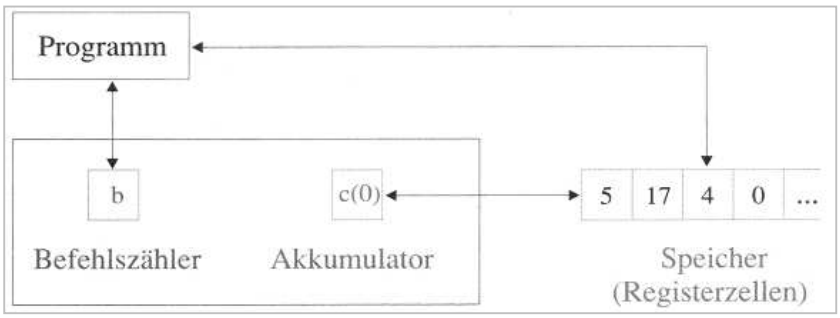


Abb: 3.1.1: Schematischer Aufbau einer Registermaschine[8]

In dieser Darstellung der Registermaschine von Asteroth/Baier wird das Programm bewusst von der Maschine getrennt um die Universalität deutlich hervorzuheben.
In welcher Art und Weise die Registermaschine das Programm speichert, wird in dieser Darstellung nicht sichtbar.

[5] [Bau2] S.198
[6] [Bau2] S.199
[7] [Ast/Bai] S.19
[8] [Ast/Bai] S.19

Man kann die Architektur der Registermaschine auch über die Struktur der informationsverarbeitenden Systeme beschreiben. Sie besteht aus:

(1) Speicher

Der **Hauptspeicher** (kurz Speicher) ist eine lineare Anordnung gleich großer Speicherzellen, deren jede eine Zahl oder Zeichenkette aufnehmen kann und eine Nummer, die sogenannte Adresse besitzt. Wir unterteilen den Speicher in Programmspeicher und Datenspeicher. Im **Programmspeicher** sind die von der Maschine auszuführenden Befehle enthalten. Der **Datenspeicher** nimmt die zu verarbeitenden Daten auf und bietet Platz für eventuelle Zwischenergebnisse. Über Datenleitungen kann mittels der Adressen auf die Speicherinhalte sowohl lesend als auch schreibend zugegriffen werden. Neben dem Hauptspeicher gibt es noch weitere Speicherbausteine, *Register* genannt, die vor allem bei der Organisation der Programmausführung mitwirken.

(2) Prozessor

Der **Hauptprozessor** (kurz Prozessor) besteht aus Rechenwerk und Steuerwerk. Letzteres, auch Leitwerk genannt, organisiert die Programmausführung. Das Rechenwerk, die arithmetisch-logische Einheit(ALU) ist ein Schaltwerk, welches auf Veranlassung des Steuerwerks sämtliche arithmetischen und logischen Operationen durchführt. Wichtiger Bestandteil des Prozessors sind die folgenden Arbeitsregister.

- Das erste Arbeitsregister ist der **Akkumulator**: Er nimmt jeweils den Inhalt einer Zelle des Datenspeichers auf; den Namen (lat.: accumulare = sammeln) hat er von seiner Funktion, die Rechenergebnisse zu sammeln. das heißt: vor Ausführung der zweistelligen Operation befindet sich der eine Operand im Datenspeicher, der andere im Akkumulator; nach der Ausführung befindet sich das Ergebnis im Akkumulator.
- Das zweite Arbeitsregister ist das **Befehlsregister** : Es nimmt jeweils den Inhalt einer Zelle des Programmspeichers, und zwar den momentan auszuführenden Befehl, auf.
- Das dritte Arbeitsregister ist das Befehlsadressregister (kurz **Befehlszähler**). Es beinhaltet jeweils eine Adresse des Programmspeichers, nämlich die des Befehls, der gerade zur Ausführung ansteht.

(3) Ein- und Ausgabeeinheit

Es wird keine besondere Ein-/Ausgabeeinheit benötigt. [9]

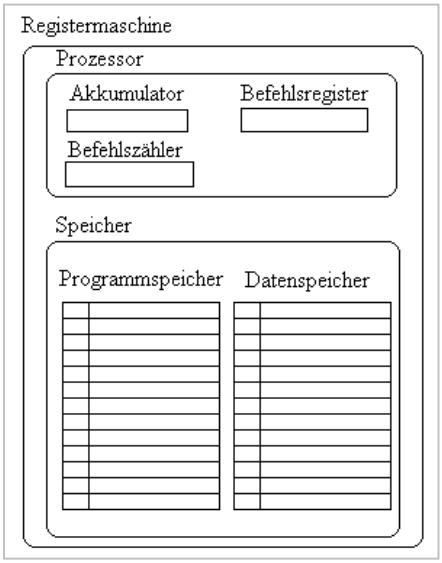


Abb: 3.1.2 schematischer Aufbau einer Registermaschine nach [Bau2]

[9] [Bau2] S.199

Diese Beschreibung von Baumann erscheint etwas umfangreich, ist aber als Grundlage für die *von-Neumann-Architektur* sehr gut geeignet. *„Die von-Neumann-Architektur stellt eine Erweiterung des theoretischen Konzepts der Registermaschinen als Grundlage der imperativen Programmierung dar.“* [10]

Neben verschiedenen Definitionen der Registermaschinen kann man zur Programmierung der Registermaschinen unterschiedliche Befehlssätze verwenden.

Asteroth und Beier geben als mögliche Befehle der Registermaschine an:
„Die Grundbefehle LOAD, STORE und die arithmetischen Operationen werden mit einem Operator *x* aufgerufen. Drei Typen von Operanden sind zulässig.

- Konstanten *#k*
- direkte Adressierung *i*
- indirekte Adressierung **i* “ [11]

Diese allgemeine Beschreibung wird in der Literatur verschieden umgesetzt.

Einige Beispiele für Befehlssätze:

Befehl	Effekt
LOAD x	$c(0) := v(x); b := b + 1;$
STORE i	$c(i) := c(0); b := b + 1;$
STORE $*i$	**IF** $c(i) \geq 1$ **THEN** $c(c(i)) := c(0); b := b + 1$ **ELSE** $b := \infty$ (* Abbruch *) **FI**
ADD x	$c(0) := c(0) + v(x); b := b + 1$
SUB x	$c(0) := \max\{0, c(0) - v(x)\}; b := b + 1$
MULT x	$c(0) := c(0) * v(x); b := b + 1$
DIV x	**IF** $v(x) > 0$ **THEN** $c(0) := c(0)$ div $v(x); b := b + 1$ **ELSE** $b := \infty$ (* Abbruch *) **FI**
GOTO j	$b := j$
JZERO j	**IF** $c(0) = 0$ **THEN** $b := j$ **ELSE** $b := b + 1$ **FI**
END	$b := \infty$ (* Das Programm wird beendet. *)

Abb: 3.1.3. Befehlssatz nach Asteroth/Baier[12]

[10] [Ho/Ker/For] S.51
[11] [Ast/Bai] S.19
[12] [Ast/Bai] S.21

Befehle - Effekt

(Transportbefehle)
load i - C[0]=C[i]; b:=b+1;
cload i - C[0]=i; b:=b+1;
iload i - C[0]=C[C[i]]; b:=b+1;
store i - b = b + 1; C[i] = C[0];
istore i - if (c(i)>=1) {b = b + 1; C[C[i]] = C[0]; } else {b = b};
(Verarbeitungsbefehle)
add i - b = b + 1; C[0] = C[0] + C[i];
cadd i - b = b + 1; C[0] = C[0] + i;
iadd i - b = b + 1; C[0] = C[0] + C[C[i]];
sub i - b = b + 1; if (C[0]>=C[i]){C[0] = C[0] - C[i];} else {C[0] = 0;}
csub i - b = b + 1; if (C[0]>=C[C[i]]){C[0] = C[0] - C[C[i]];} else {C[0] = 0;}
isub i - b = b + 1; if (C[0]>=i){C[0] = C[0] - i;} else {C[0] = 0;}
mult i - b = b + 1; C[0] = C[0] * C[i];
cmult i - b = b + 1; C[0] = C[0] * i;
imult i - b = b + 1; C[0] = C[0] * C[C[i]];
div i - if (C[i]>0) {C[0] = C[0] / C[i]; b = b + 1;} else (b = b);
cdiv i - if (i>0) {C[0] = C[0] / i; b = b + 1;} else (b = b);
idiv i - if (C[C[i]]>0) {C[0] = C[0] / C[C[i]]; b = b + 1;} else (b = b);
(Sprungbefehle)
goto i - b = i;
igoto i - b = C[i];
jzero i - if (C[0]==0) b = i; else b = b+1;
(Terminierung)
end - Programmende

Bei diesem Befehlssatz werden die unterschiedlichen Bedeutungen der Operanden von Asteroth/Baier durch verschiedene Befehle übernommen und die Operanden sind, je nach Befehl, Adresse oder Konstante.
In verschiedenen anderen Befehlssätzen, z.B. bei Baumann, wird auf die indirekte Adressierung verzichtet und die Operanden sind bis auf eine Ausnahme durchweg Adressen. Die Ausnahme bildet der Befehl zum Laden einer Konstanten in den Akkumulator.
Es ist zu beachten, dass sich in beiden angegebenen Befehlssätzen die Inhalte des Datenspeichers natürliche Zahlen sind. Es können aber mit Hilfe der Registermaschine auch ganze oder rationale Zahlen, Graphen oder lineare Listen bearbeitet werden. Näheres dazu in Asteroth/Baier [Ast/Bai] Seite 24 ff.

Mit diesen Überlegungen sind die Architektur und möglichen Befehlsätze von Registermaschinen dargestellt. Worüber noch keine Aussagen getroffen worden sind, ist das Eingabe- und Ausgabeverhalten von Registermaschinen.
Dazu muss man Eingaberegister und Ausgaberegister festlegen. Nach Asteroth/Baier sollte die Eingabe in den ersten Registerzellen 1,2,3,..,k des Datenspeichers stehen und die Ausgabe in einer festgelegten Registerzelle i. Meist wird als Ausgabezelle der Akkumulator (i = 0) verwendet.

Damit ist die durch die Registermaschine $\Re$ berechnete partielle Funktion $f_{\Re} : IN^k \to IN$ gegeben durch:

- $f_{\Re}(n_1,...,n_k) = \perp$, falls $\Re$ für die initiale Registerbelegung $c[n_1,...,n_k]$ nicht terminiert.
- $f_{\Re}(n_1,...,n_k) = c(i)$, falls $\Re$ für die initiale Registerbelegung $c[n_1,...,n_k]$ mit der Registerbelegung c terminiert.

Dabei ist k die Anzahl der Eingaberegister und Register i das Ausgaberegister von $\Re$. [13]

[13] [Ast/Bai] S.23

Die Programmausführung durch die Registermaschine kann in 5 Schritten, dem **Befehlszyklus**, beschrieben werden.

Baumann stellt den **Befehlszyklus** wie folgt dar.

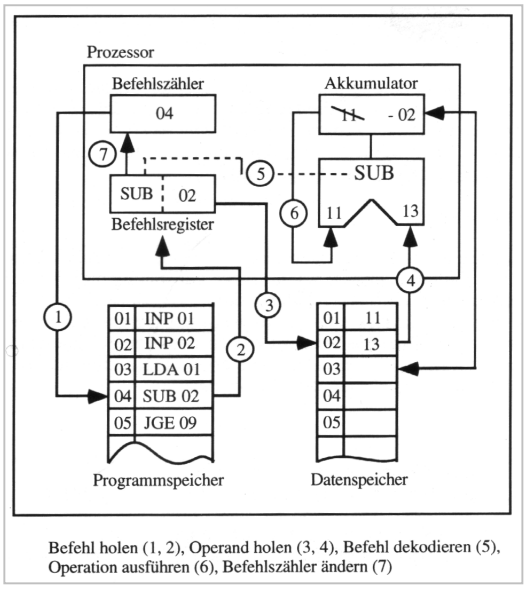


Abb: 3.1.4 Befehlszyklus einer Registermaschine [14]

1. **Befehl holen (1,2)**
   Aus dem Programmspeicher wird das Befehlswort (Befehl und Operand) in das Befehlsregister geladen, welches die Adresse des Wertes im Befehlszählers besitzt.
2. **Operand holen (3,4)**
   Ist der Operand des Befehlswortes im Befehlsregister eine Adresse so wird mit dieser Adresse der Operand für die Operation aus dem Datenspeicher in die ALU geladen. Ansonsten wird der Operand des Befehlswortes in die ALU geladen.
3. **Befehl dekodieren (5)**
   Der ALU wird mitgeteilt, welche Operation ausgeführt werden soll.
4. **Operation ausführen (6)**
   Die ALU holt den Wert aus dem Akkumulator und führt die Operation durch. Das Ergebnis steht anschließend im Akkumulator.
5. **Befehlszähler ändern (7)**
   Nach der Durchführung des Befehls wird der Befehlszähler in der Regel um den Wert 1 erhöht. Ausnahmen bilden die Sprungbefehle. In diesen Fällen wird die Sprungadresse des Befehls an den Befehlszähler übergeben.

Somit wäre der prinzipielle Aufbau und die Funktionsweise der Registermaschine beschrieben.
Auf die Bedeutung der Registermaschine für die theoretische Informatik in Bezug auf Berechenbarkeit, Algorithmenbegriff oder Kostenmaße gehe ich an dieser Stelle nicht ein, da der Schwerpunkt der Arbeit auf der objektorientierten Programmierung liegt.

[14] [Bau2] S.201

3.2. Grundkonzepte des objektorientierten Softwareentwicklung

3.2.1. Grundbegriffe und Prinzipien der Objektorientierung

Das Gerüst der objektorientierten Programmierung bilden **Objekte** und Objekttypen (**Klassen**), die eigene **Attribute** und **Methoden** besitzen und die Fähigkeit haben, über **Nachrichten** miteinander zu kommunizieren. Die grundlegenden Prinzipien der objektorientierten Programmierung sind **Abstraktion**, **Kapselung und Modularisierung**.

Objekte sind programmtechnisch Datenstrukturen, welche Objekte der Realität mit ihren spezifischen Eigenschaften beschreiben. Dabei werden die Objekte der Realität so modelliert, dass unwichtige Details vernachlässigt werden und sich das Modell nur auf die wesentlichen Merkmale beschränkt. Dieser Vorgang wird gemeinhin als **Abstraktion** bezeichnet.
Alle Objekte mit gleichen Eigenschaften werden in einem Objekttyp (**Klassen)** zusammengefasst. Diese **Klassen** definieren die **Attribute** und **Methoden** der von ihr erzeugten **Objekte** (Instanzen).

Als **Zustand** von Objekten oder Klassen bezeichnet man die statischen Eigenschaften (**Attribute**). Die Attributwerte der Objekte können durch dynamische Eigenschaften (**Methoden**) verändert werden um die Modellierung der realen Umwelt an veränderte Bedingungen anzupassen. Diese Methoden charakterisieren das **Verhalten** der Objekte.

Jedes Objekt ist durch seinen Namen, den Zustand und sein Verhalten vollständig beschrieben. Diese drei Merkmale bezeichnet man als **Kapselung**.
Dies stellt einen wesentlichen Unterschied zur prozeduralen Programmierung dar, in der Daten und Funktionen getrennt sind.
Dieses Prinzip der **Kapselung** aller nötigen Informationen in einem Objekt fördert ein weiteres wichtiges Prinzip der objektorientierten Programmierung, die **Modularisierung**.
Das Prinzip der **Modularisierung** fördert und fordert die Mehrfach- und Wiederverwendbarkeit, präzise und vollständige Schnittstellenbeschreibungen, das Geheimnisprinzip und die Verständlichkeit, Überschaubarkeit und Wartbarkeit der Programmierergebnisse. [15]

Ein wichtiger Punkt der objektorientierten Programmierung ist die Kommunikation zwischen den Objekten. Durch die Kapselung und das Geheimnisprinzip (es sind nur die öffentlichen Schnittstellen der Objekte nach außen bekannt, die Attribute und Methodenspezifikationen sind „versteckt“) können nicht wahllos Zugriffe auf Attribute und Methoden der einzelnen Objekte durchgeführt werden. Objekte kommunizieren durch den Austausch von **Nachrichten**. Ein Objekt sendet eine Nachricht an ein anderes Objekt. Dabei besteht die Nachricht aus erforderlichen Parametern und einer öffentlichen Methode des Empfängers. Der Empfänger ist für die Realisierung dieser Methode allein verantwortlich, da die Realisierung dem Sender verborgen ist. Dadurch können aus einem Objekt (öffentliche) Methoden anderer Objekte aufgerufen werden.

Damit die Objekte miteinander Nachrichten austauschen können, müssen sie sich „kennen“, in Beziehung zueinander stehen. Diese Beziehungen zwischen Klassen oder ihren Instanzen werden durch **Assoziationen** („KENNT-Beziehung“) beschrieben. „*Durch die Verbindung*

[15] [Erler] S.21

über eine Assoziation können die Objekte der verbundenen Klassen miteinander kommunizieren. Es besteht dann auf der Ebene der Objekte eine Instanzenverbindung (link)“.[16]
Die Assoziationen sind dadurch gekennzeichnet, wie viele Klassen miteinander in Beziehung stehen. Man unterscheidet binäre, reflexive und N-äre Assoziationen.

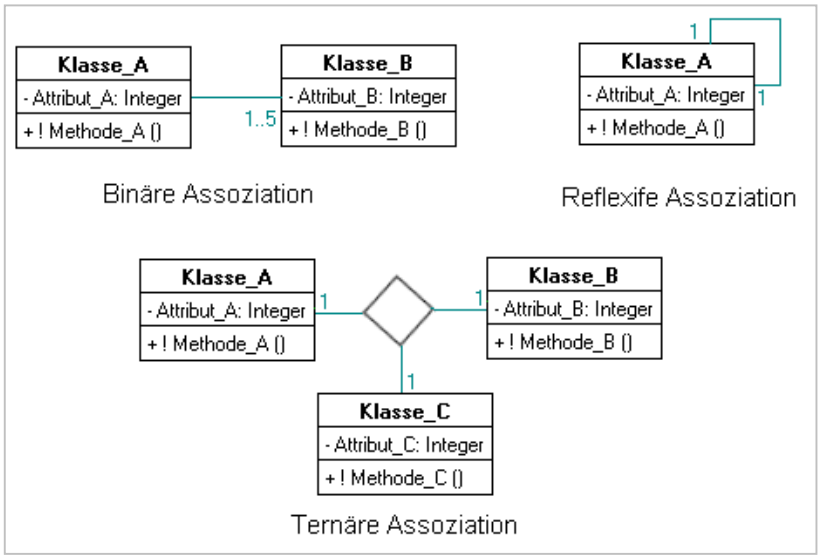


Abb.3.2.1.1. UML-Darstellung verschiedener Assoziationen

Eine besondere Form der Assoziation ist die **Aggregation**. Es handelt sich dabei um Beziehungen, bei denen sich Objekte einer Klasse aus Objekten einer oder mehrerer Klassen zusammensetzen (HAT-Beziehung). Dadurch wird eine Hierarchie der Objekte definiert. Oft wird die Aggregation auch als Gesamt-Teil-Struktur bezeichnet.

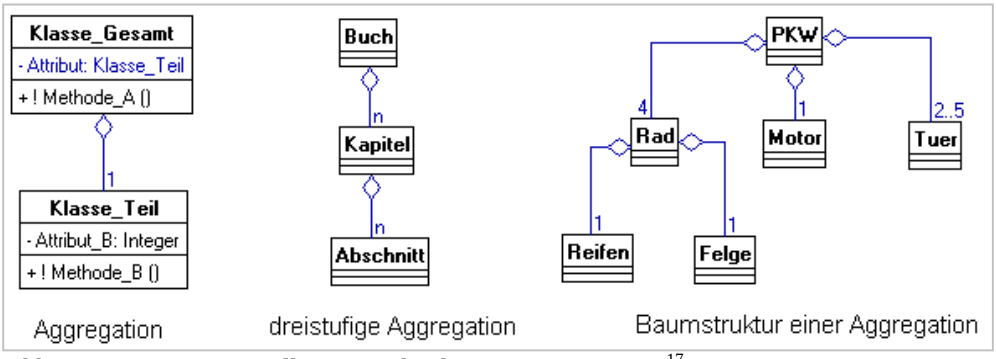


Abb: 3.2.1.2 UML-Darstellung verschiedener Aggregationen [17]

Auf weitere wichtige Begriffe der objektorientierten Programmierung wie Vererbung, Polymorphie und dynamisches Binden gehe ich an dieser Stelle nicht ein, da sie für diese Arbeit nicht relevant sind.

[16] [Erler] S.79
[17] nach [Erler] S.87 ff

3.2.2. Phasen der Softwareentwicklung

Ein wesentliches Ziel der Softwareentwicklung ist die Produktion qualitativ hochwertiger Software.
„In Anlehnung an DIN 55350 ist Softwarequalität die Gesamtheit der Eigenschaften oder Merkmale, welche ein Produkt (die Software) in Verwendung und (Weiter-)Entwicklung aufweist, um die gegebenen Anforderungen zu erfüllen.
Um diese relativ abstrakte Definition in Bezug auf Software mit Leben zu füllen, kann zunächst eine grobe Unterscheidung in die Produktqualität und die Gebrauchsqualität von Software getroffen werden.“ [18]

Bei der Produktqualität geht es um innere Qualitätsmerkmale der Software wie **Verständlichkeit**, **Änderbarkeit** und **Wiederverwendbarkeit**. Die Gebrauchsqualität zielt auf äußere Qualitätsmerkmale wie z. B. **Korrektheit, Zuverlässigkeit**, **Effizienz** und **Fehlerrobustheit**.

Um die Qualitätsmerkmale zu erfüllen kann man die Softwareentwicklung in verschiedene Phasen unterteilen.

- Anforderungsermittlung oder Problemanalyse
- Entwurf und Modellierung
- Implementation
- Test und Revision

In der heutigen Zeit werden dafür objektorientierte Methoden bevorzugt. Diese Methoden sind:

- objektorientierte Analyse (OOA)
- objektorientiertes Design (OOD)
- objektorientierte Programmierung (OOP) [19]

Bei der **objektorientierten Analyse** werden die Anforderungen an das Softwaresystem entwickelt und in einer Anforderungsspezifikation (Pflichtenheft) festgehalten. Die Endabnahme der Software erfolgt nach den Vorgaben der Anforderungsspezifikation.
Dazu wird unabhängig von der späteren Implementierung das zu lösende Problem durch Objekte und Klassen sowie ihren Beziehungen beschrieben. Dabei geht es um die Abstraktion der Realität und die Kapselung in Objekten und Klassen des objektorientierten Modells.
Im **objektorientierten Design** wird auf der Grundlage der objektorientierten Analyse das erstellte Modell an die technische Plattform angepasst und um die Modellierung der Benutzeroberfläche und die Gestaltung der Datenhaltung erweitert.
Zur Notation der Darstellung der Modellierungen von Analyse und Design eignet sich UML (Unified Modeling Language).
Die Implementierung des Modells erfolgt in einer objektorientierten Programmiersprache.
Durch den strengen modularen Aufbau können entworfene Klassen direkt implementiert und getestet werden. [20]

[18] [Six/Winter] S.10
[19] [Erler] S.30
[20] nach [Erler] S.30

Die Ablauforganisation der Entwicklungsschritte beschreiben verschiedene Vorgehensmodelle. *„Sie zerlegen den Entwicklungsprozess gewöhnlich in zeitlich aufeinanderfolgende Phasen, die jeweils ein Zeitintervall mit den darin stattfindenden Aktivitäten beschreiben.“* [21]

Ein erstes und bis heute eingesetztes Modell ist das **Wasserfallmodell**. Durch die klar abgegrenzten und aufeinanderfolgen Phasen ergeben sich Vorteile für das Management, Planung und die Kontrolle der Softwareentwicklung. Da in diesem Modell die Wiederaufnahme von Aktivitäten aus früheren Phasen nicht vorgesehen ist, ergibt sich ein Problem.
„Jede Anforderungsspezifikation, jede Entwurfsspezifikation, jede Implementierung enthält Fehler, die in darauf aufbauenden Aktivitäten entdeckt werden und deren Korrektur zu rückwirkenden Änderungen führt.“[22]

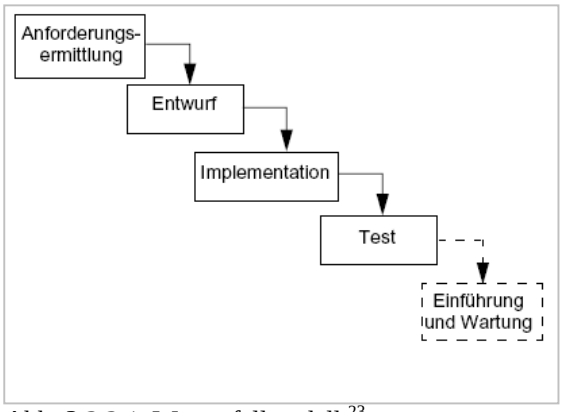


Abb: 3.2.2.1. Wasserfallmodell [23]

Besser geeignet sind iterative oder evolutionäre Modelle, in denen die Abfolge der Phasen unterbrochen werden können und, falls nötig, Sprünge z.B. aus der Implementation ins Design oder aus der Analyse in die Programmierung möglich sind.[24]

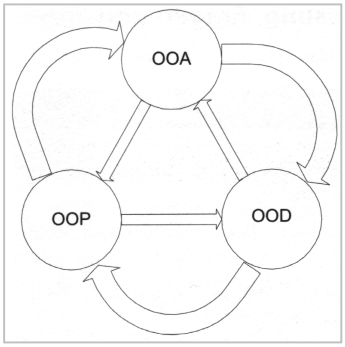


Abb: 3.2.2.2. evolutionäres Vorgehensmodell [25]

Weitere Vorgehensmodelle wie Rational Unified Process, Prototyping oder Extreme Programming sind für die Schule nicht relevant.

[21] [Six/Winter] S.20
[22] [Six/Winter] S.22
[23] [Six/Winter] S.21
[24] [Erler] S.31
[25] [Erler] S.31

4. Unterrichtliche Umsetzung

Die Programmierung der Registermaschine unter Berücksichtigung objektorientierter Gesichtspunkte soll, wie in 2.1. erläutert mit den dort formulierten Voraussetzungen in der 12/I durchgeführt werden. Die Ausführungen sind auf einen Grundkurs in der Übungs- und Festigungsphase des Themas objektorientierte Softwareentwicklung ausgerichtet, können aber auch in einem Leistungskurs, eventuell auch in der Erarbeitungsphase des Themas, eingesetzt werden.

4.1. allgemeine Überlegungen

4.1.1. Ziele

Das Hauptziel ist die Entwicklung der **Handlungskompetenz** der Schüler und Schülerinnen, welche sich in den Dimensionen von Fachkompetenz, Personalkompetenz und Sozialkompetenz entfalten. (KMK, 2000)

Fachkompetenz:
Die Schüler und Schülerinnen sollen in dieser Unterrichtseinheit

- die grundlegenden Methoden der objektorientierten Softwareentwicklung kennen.
- diese auf gestellte Probleme anwenden können.
- können Problemlösestrategien entwickeln
- die Entwicklung von Informatiksystemen verstehen.
- die Registermaschine als Modell eines programmgesteuerten Automaten kennen.
- ihren Aufbau und ihre Funktionsweise verstehen.
- entwickelte Modelle in einer objektorientierten Programmiersprache implementieren können.

Personal- und Sozialkompetenz:
Neben diesen kognitiven Zielen sollen aber auch die sozialen Kompetenzen der Schüler und Schülerinnen gefordert und gefördert werden.
Dazu gehören:

- Teamfähigkeit
- Verantwortungsbewusstsein
- Selbständigkeit
- Eigeninitiative
- Stärkung des Selbstbewusstseins
- Kommunikationsfähigkeit
- Kreativität

Die Ziele der Personal- und Sozialkompetenz sind für alle Phasen relevant. Die kognitiven Ziele werde ich in den einzelnen Phasen noch weiter spezifizieren.

4.1.2. methodisch-didaktische Überlegungen

Mit der Auswahl des Themas „objektorientierte Softwareentwicklung“ konzentriert sich die Unterrichtseinheit auf einen Bildungskern des Schulfaches Informatik. Aus dem Softwareentwicklungsprozess wurde das Konzept der „Fundamentalen Ideen der Informatik“(Schwill,1993) exemplarisch begründet und abgeleitet. Die Softwareentwicklung

stellt das allgemein bildende Kernstück der Fachwissenschaft Informatik dar und ist deshalb zur Strukturierung des gleichnamigen Schulfaches geeignet. [26]

Dieses Thema

- erstreckt sich über mehrere Schuljahre,
- ordnet den Stoff vom Einfachen zum Komplizierten (Spiralcurriculum)
- fördert die Vernetzung der Unterrichtsinhalte.
- beinhaltet die Masterideen der „Fundamentalen Ideen“;
  Algorithmisierung, **strukturierte Zerlegung** und **Sprache** [27]

und ist somit als Unterrichtsthema höchst relevant.

Bei der Auswahl der Registermaschine als Programmierinhalt stand im Vordergrund, dass dieses Problem den folgenden Kriterien gerecht wird:

- Realitätsbezug
  Die Programmierung einer Simulationsumgebung repräsentiert einen exemplarischen Ausschnitt der Wirklichkeit und ist für den Einsatz typisch.
- Realisierbarkeit
  Der Schwierigkeitsgrad ist den Kenntnissen der Schüler und Schülerinnen anpassbar, der zeitliche Rahmen ist abschätzbar.
- Vorarbeiten
  Der Umfang der Vorarbeiten ist relativ gering.
- Modularisierbarkeit
  Das Problem ist sehr gut modularisierbar, wobei die Einzelbausteine hinreichend komplex sind.
- Reduzierbarkeit / Erweiterbarkeit [28]
  Das Thema ist je nach Bedarf durch Vorgabe einzelner Module reduzierbar oder durch Erweiterung des Funktionsumfangs der Registermaschine oder der Benutzeroberfläche leicht ausbaufähig.

Nach den didaktischen Überlegungen stellt sich die Frage nach einer geeigneten Methode die formulierten Ziele zu erreichen. Um den aktiven Schüler bzw. die aktive Schülerinnen in den Mittelpunkt des Unterrichts zu stellen bietet sich für diese Unterrichtseinheit der Projektunterricht an. Der Projektunterricht ist eine Mischung der Unterrichtsformen.

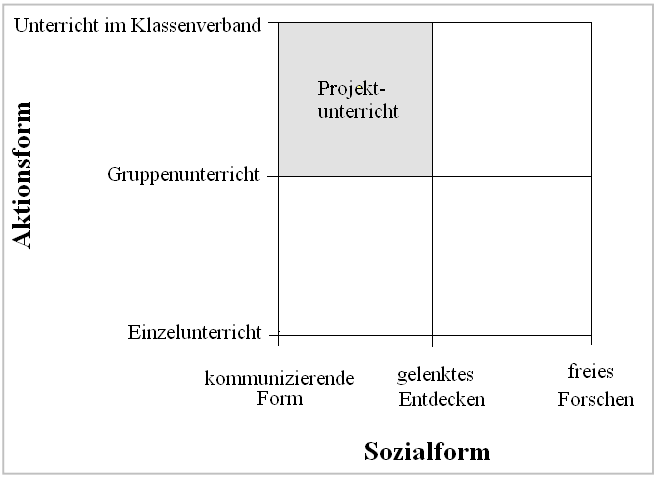


Abb.: 4.1.2.1. Einordnung des Projektunterrichts [29]

[26] [Schw/Bai] S.44
[27] vergl.[Schw/Bai] S.89
[28] vergl. [Röhner] S.6

Der Projektunterricht als handlungsorientierter Unterricht bietet große Möglichkeiten der Kommunikation der Schüler untereinander sowie Freiraum für eigene Aktivitäten.
„Der Arbeits- und Lernprozess, der durch die Projektidee ausgelöst und organisiert wird, ist dabei ebenso wichtig wie das Handlungsergebnis oder Produkt, das am Ende des Projektes stehen soll. Projekte eröffnen die Chance, die gesellschaftlich vorgegebene Trennung von Kopf- und Handarbeit ein Stück weit aufzuheben.“ [30]
Während der einzelnen Projektphasen werden die Sozial- und Aktionsformen variiert. Dies führt zur Förderung der kooperativen Arbeit und der Konfliktlösung während des gesamten Projektverlaufs. Darauf gehe ich in den methodisch-didaktischen Überlegungen zu den einzelnen Phasen genauer ein.
Für dieses Projekt habe ich geplant, nicht, wie üblich, jede Gruppe ein zu entwickelndes Modul erstellen zu lassen, sondern jede Gruppe entwickelt seine eigene Registermaschinensimulation. In der Gruppe werden Modulspezialisten benannt, die sich in der Designphase zu Expertengruppen zusammenschließen, um anschließend die Implementation Ihres Moduls in Ihrer Gruppe selbstständig vorzunehmen. Damit entsteht folgende Abfolge der Sozialformen für die einzelnen Phasen:

Phase		Sozialform
Einführung		
	Erarbeitungsphase	in der Gruppe
	Ergebnispräsentation	im Klassen- / Kursverband
Analysephase		
	Erarbeitungsphase	in der Gruppe
	Ergebnispräsentation	im Klassen- / Kursverband
Designphase		
	Erarbeitungsphase	in Expertengruppen
	Ergebnispräsentation	in der Gruppe
Implementation		
	Programmierungsphase	Einzelarbeit (Partnerarbeit)
	Modulzusammenführung	in der Gruppe
Projektpräsentation		im Klassen- / Kursverband

Ein weiterer wichtiger Punkt neben den Sozial- und Aktionsformen bildet die Frage, wie man möglichst alle Projektgruppenmitglieder möglichst gleichberechtigt in die Arbeit der Gruppe integriert. Dazu sollte während der gesamten Projektlaufzeit die Schüler und Schülerinnen im Wechsel unterschiedliche Aufgaben übernehmen. Neben ihren Aufgaben als Modulexperten könnte dies sein

- Projektüberwacher
- Schnittstellenbeauftragter
- Tester
- Dokumenteur
- Kümmerer [31]

[29] vergl. [Schw/Bai] S.297
[30] [Röhner] S.1
[31] [Röhner] S.5

4.2. Einführungsphase

4.2.1. Ziele

In der Einführungsphase sollen die Schüler und Schülerinnen sich mit der Registermaschine vertraut machen.

Sie

- kennen die grundlegenden Komponenten der Registermaschine und ihre Funktionen. (Datenspeicher, Programmspeicher, Akkumulator, Befehlsregister, Befehlszähler)
- kennen den Befehlssatz der Registermaschine und kennen die Bedeutung der einzelnen Befehle.
- verstehen das Prinzip der direkten und indirekten Adressierung.
- kennen den Befehlszyklus der Registermaschine.
- können Registermaschinenprogramme interpretieren.

4.2.2. methodisch-didaktische Überlegungen

Gemäß der Aufgabenstellung (siehe Anhang) beginnt die Unterrichtseinheit mit der Bearbeitung der Aufgabenstellung 1. Dazu erhalten die Schüler und Schülerinnen Arbeitsmaterial (siehe Anhang), welches sich grundsätzlich an den Vorstellungen von Baumann[32] orientiert. Diese strukturierte Darstellung, welche sich stark auf die Komponenten der Registermaschine bezieht, ist für die Abstraktion in der Analysephase sehr geeignet. Der von Baumann angegebene Befehlssatz (Assemblersprache) beinhaltet nur Operationen mit direkter Adressierung. Dies ist für das Verständnis der Registermaschine und die spätere Implementation von Vorteil, verkompliziert aber die spätere Erstellung eigener Registermaschinenprogramme. Ich habe mich bei der Auswahl des Befehlssatzes durch Ausführungen von Asteroth/Baier über Operanden mit direkter und indirekter Adressierung und konstanten Operanden leiten lassen. Dabei unterscheide ich diese aber nicht durch Präfixe für die Operanden (i - direkt, *i - indirekt , #i - konstant), sondern durch Präfixe für die Befehle (add i - direkt, iadd i - indirekt, cadd - konstant). Dies ist aus meiner Sicht für die Schüler und Schülerinnen leichter zu implementieren. Der Befehlszyklus der Registermaschine wird im Material in Wort und Bild beschrieben und an einem einfachen Beispiel, der Maximumbestimmung, in einer Programmablauftabelle dargestellt.

Die Schüler und Schülerinnen werden für das ganze Projekt in Gruppen zu maximal vier Personen aufgeteilt. Dabei wäre es von Vorteil gleichstarke Gruppen zu bilden. Ich gehe davon aus, dass die Größe eines Informatikkurses 16 selten überschreiten wird und so selten mehr als 4-5 Gruppen entstehen.
In dieser Phase sollte die beherrschende Unterrichtsform die Gruppenarbeit sein, in der sich die Kommunikation vorwiegend innerhalb der Gruppe vollzieht. Dabei können auftretende Probleme innerhalb der Gruppe diskutiert und gelöst werden. Dies ist eine wichtige Methode um die in 4.1.1. dargestellten Ziele der Persönlichkeits- und Sozialkompetenz zu erreichen. Der Lehrer bietet in dieser Phase nur Unterstützung an, wenn in den Gruppen Probleme nicht gelöst werden können und absehbar ist, dass die zeitlichen Vorgaben nicht eingehalten werden können.

[32] [Bau02] S.199 ff

Zur Lernzielkontrolle sollen die Gruppen das Arbeitsblatt I (siehe Anhang) bearbeiten und die Ergebnisse mit einer Musterlösung (siehe Anhang) vergleichen. Dadurch wird die Kommunikation in der Gruppe gefördert.
Es ist darauf zu achten, dass alle Schüler in dieser Phase in der Gruppe aktiv beteiligt sind. Ziel sollte es sein, dass jeder Schüler der Gruppe das Gruppenergebnis vorstellen kann. Bei der Programmentwicklung für das Beispiel n^m können durchaus mehrere Lösungen durch die Gruppen erarbeitet worden sein und sollten auch von mehreren Schülern bzw. Schülerinnen anschließend im Kursverband vorgestellt und diskutiert werden.
Damit wird gesichert, dass alle Schüler und Schülerinnen am Ende der Einführungsphase das notwendige Wissen für die weiteren Projektphasen besitzen.

4.2.3. Verlaufsplanung

geschätzter Zeitaufwand: 1 Doppelstunde

Inhalt	Sozialform	Schüleraktivität	Lehreraktivität
Gruppenbildung			
Aufgaben-stellung	Unterrichtsge-spräch	▫ Verständnisprobleme klären	▫ erläutert Aufgaben-stellung
Materialerar-beitung	Gruppenarbeit	▫ lesen ▫ Beispiel nachvollziehen ▫ Verständnisprobleme klären	▫ Hilfestellung
Bearbeitung des Arbeitsblattes	Gruppenarbeit	▫ Transfer des Beispiels auf neue Problemstel-lungen ▫ Ergebnisdiskussion ▫ Lösungsvergleich	▫ Hilfestellung
Ergebnisvor-stellung	Schülervortrag Unterrichtsge-spräch	▫ einzelne Schüler stellen die Ergebnisse vor ▫ andere moderieren ▫ Ergebnisdiskussion zwischen den Gruppen	▫ wenn nötig Mode-rator

4.3. Analysephase

4.3.1. Ziele

In dieser Phase vertiefen und festigen die Schüler ihr Wissen über die Analysephase der objektorientierten Softwareentwicklung.
Sie :

- kennen des Konzept der Abstraktion und können es auf das gegebene Beispiel anwenden.
- können die Grundbegriffe der objektorientierten Programmierung anwenden.
- können das Konzept der Modularisierung anwenden.
- kennen ausgewählte Elemente der Unified Modeling Language (UML)

4.3.2. methodisch-didaktische Überlegungen

Ziel des Analyseprozesses ist es, *„ein System von Objekten zu finden und zu arrangieren, die im gemeinsamen Zusammenspiel das reale System nachbilden und die gestellte Aufgabe mit verteilten Verantwortlichkeiten erledigen.“* [33]
Dazu muss in der Anforderungsdefinition (siehe Anlage) klar formuliert werden, welche Aufgabe das System erledigen muss. Die Schüler und Schülerinnen müssen erkennen, dass diese eine Beschreibung der Funktionalität, des Leistungsumfangs und des Verhaltens nach außen beinhaltet, welche aus den angegebenen Materialien erarbeitet werden muss.

Die Abstraktionsphase untergliedert sich daraufhin in drei Schritte.

1. Bestimmung der Objekte / Klassen des Systems
2. Bestimmung der Verantwortlichkeiten der gefundenen Objekte / Klassen
3. Bestimmung der Verbindungen zwischen den Objekten / Klassen

Es ist wichtig, dass die Schüler diese Abstraktionsschritte kennen, da dies, neben der Modellierung, die entscheidende Grundlage der Softwareentwicklung darstellt. Diese Schritte kann man als Leitlinien plakativ im Raum für alle sichtbar anbringen oder als Anhang zur Aufgabenstellung mitgeben. Dabei ist es von Vorteil, wenn man den Schülern und Schülerinnen zu den einzelnen Schritten Fragen formuliert, deren Beantwortung ihnen in den einzelnen Schritten Lösungen aufzeigt.
Die Vorschläge von Spolwig sind dazu sehr gut geeignet.

1. Welche Objekte gibt es im System ? *(Objektnamen)* und welche Informationen muss sich das Objekt merken ? *(Attribute)*
2. Was tun sie, was kann man mit ihnen tun ? *(Methoden)*
3. Welche Objekte stehen in welcher Beziehung zueinander? *(Assoziationen)*

In unserem Fall bietet sich die Textanalyse der Anforderungsspezifikation an, da die Betrachtung einer realen Registermaschine nicht möglich ist.
Dabei wird der Text auf Substantive als Kandidaten für Objekte und auf Verben als Kandidaten für Operationen untersucht.

[33] [Spolwig] S.31

Objekte können sein:
- konkrete Gegenstände
- Organisationseinheiten
- Rollen von Personen
- Geräte oder Geräteteile
- Funktionalitäten in der Benutzeroberfläche

Die Registermaschine ist für diese Analysephase sehr gut geeignet, da die Objekte sehr gut erkennbar sind.
Potentielle Objektkandidaten sind:
- Registermaschine
- Programmspeicher
- Datenspeicher
- Prozessor
- Akkumulator
- Befehlsregister
- Befehlszähler
- ALU

Als nächstes ist zu klären, welche Attribute sie besitzen.
„Eine gewisse Schwierigkeit liegt darin, dass mitunter unklar ist, ob man es mit einem Attribut oder vielleicht sogar mit einem eigenständigen Objekt zu tun hat. Dafür bleibt am Ende nur der Trost – man muss sich nachvollziehbar entscheiden.“ [34]

In unserem Fall kann man bei dieser Arbeit sogar schon eine Hierarchie erstellen.
Die Registermaschine ist ein Objekt mit den Attributen Programmspeicher, Datenspeicher und Prozessor, die wiederum eigene Objekte darstellen.
Das Objekt Prozessor besitzt die Attribute Akkumulator, Befehlsregister, Befehlszähler und ALU. In der späteren Modellierungsphase werden wir sehen, dass für diese Attribute geeignete Datentypen bereitstehen und keine eigenen Klassen definiert werden müssen.

Es bleiben also übrig **Registermaschine**, **Datenspeicher**, **Programmspeicher**, **Prozessor.**

Der Datenspeicher und der Programmspeicher besitzen als Attribut adressierbare Speicherzellen.

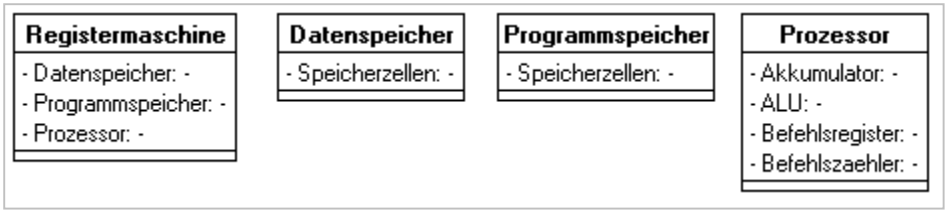


Abb.: 4.3.2.1. UML-Diagramm Objekte der Registermaschine (mit Attributen)

[34] [Spolwig] S.33

Bei der Bestimmung der Verantwortlichkeiten muss geklärt werden, welche Aufgaben und Dienste die Objekte übernehmen.
Diese Verantwortlichkeiten werden als Methoden der Objekte festgehalten.

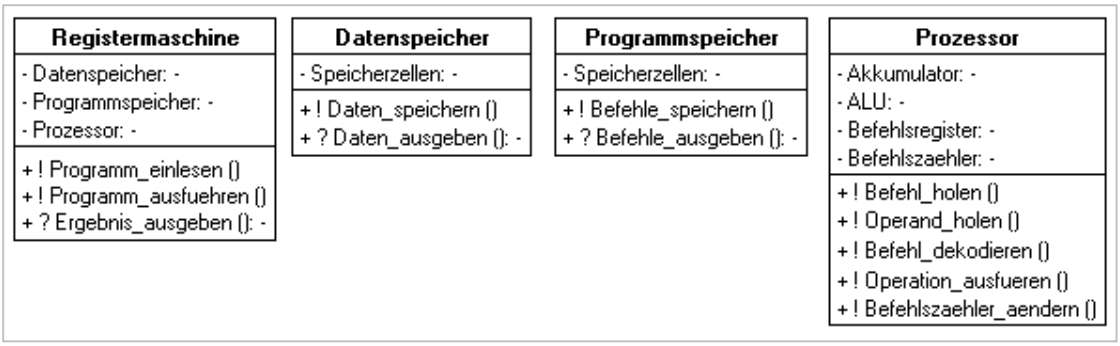


Abb.: 4.3.2.2. UML-Diagramm Objekte der Registermaschine (mit Methoden)

Bei der Formulierung der Verantwortlichkeiten kann man in dieser Phase auch sehr gut umgangssprachlich vorgehen. Dies ist besonders bei komplexeren Verantwortlichkeiten von Vorteil. z.B.:

- *Berechnen der Raumdiagonale eines Quaders* oder
- *Liefert den Wert des letzten Listenelements*

In unserem Fall kann man die Verantwortlichkeiten schon in Form von Methodennamen formulieren.

Als letzter Schritt der Analysephase muss die Verknüpfung zwischen den Objekten bzw. Klassen bestimmt werden.
Dabei stellt man fest, dass zwischen dem Objekt Registermaschine und den Objekten Datenspeicher, Programmspeicher und Prozessor eine Aggregation vorliegt. Für die Schüler kann man für diese Gesamt-Teil-Struktur auch den Namen „**HAT**-Beziehung“[35] prägen.
Dabei kann man die umgangssprachliche Formulierung: „Die Registermaschine **HAT** einen Prozessor, einen Daten- und einen Programmspeicher“ nutzen. Dies kommt den Schülern bei der Bestimmung der Arten der Beziehungen entgegen.
Außerdem muss eine Beziehung zwischen Prozessor und Datenspeicher (schreiben und lesen) und Prozessor und Programmspeicher (lesen) bestehen. Diese Beziehungen sind Assoziationen, da sie die Kommunikation zwischen diesen Objekten ermöglichen. Man kann sie auch als „**KENNT**-Beziehung“[36] betrachten. „Der Prozessor **KENNT** den Daten- und den Programmspeicher, da er Daten speichern bzw. lesen und Befehle lesen können muss.“

Damit steht am Ende dieser Analysephase ein Abstraktionsmodell einer Registermaschine. *„Diese gefundene Klassen- bzw. Objektstruktur wird sicherlich nicht das endgültige Ergebnis sein. Im Verlauf der Modellierungsphase wird es immer Änderungen aus neu gewonnen Sichtweisen geben.“* [37]

[35] aus dem Case-Tool UMLed
[36] aus dem Case-Tool UMLed
[37] [Spolwig] S.33

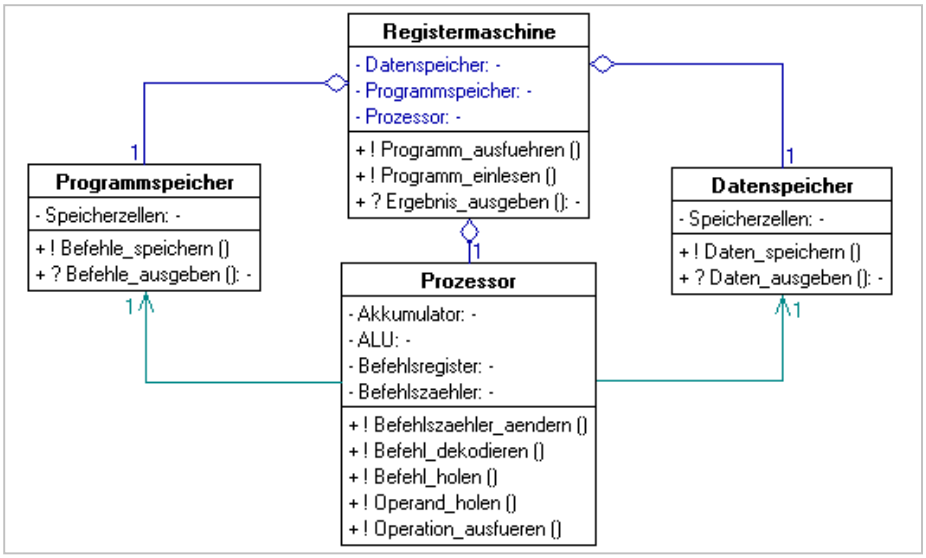


Abb.: 4.3.2.3. UML-Diagramm Objekte der Registermaschine und ihre Beziehungen

Zur Darstellung des Analyseergebnisses und als Grundlage für die Designphase eignet sich die Darstellung der Klassenstruktur in einem UML – Klassendiagramm.

Wie unter 4.1.2. beschrieben wird diese Phase in den einzelnen Gruppen bearbeit, wobei jeder Schüler bzw. jede Schülerinnen seine eigenen Abstraktionsvorstellungen einbringen, begründen und gegebenenfalls modifizieren muss, um am Ende ein Modell als Gruppenergebnis vorstellen zu können. Dies fordert und fördert bei den Schülern und Schülerinnen Kommunikationsfähigkeit, Selbstbewusstsein, Kritikfähigkeit, Engagement und Teamfähigkeit. Diese sozialen Kompetenzen stehen durch die gewählte Methode im Zentrum der Phase.
Jede Gruppe wird verschiedene Ergebnisse erarbeitet haben, welche in einer anschließenden Präsentationsphase im Klassen- oder Kursverband vorgestellt und verteidigt werden. Für den weiteren Verlauf gibt es an dieser Stelle zwei Möglichkeiten. Die eine Möglichkeit ist; jede Gruppe arbeitet mit ihrem Modell weiter oder alle Gruppen einigen sich auf ein Modell.
Beide Möglichkeiten haben Vor- und Nachteile.
Der natürliche Weg wäre dass jede Gruppe mit ihren ureigenen Überlegungen weiterarbeitet. Dies fördert die Kreativität und das Selbstbewusstsein der einzelnen Gruppen. Grundlage dafür ist, dass im Klassenverband abgesichert wird, dass alle Modelle keine grundsätzlichen Fehler beinhalten. Als Nachteil für diesen Weg sehe ich zum einen, dass der Lehrer sehr flexibel im weiteren Projektverlauf bei auftretenden Problemen reagieren muss und bei mehreren Gruppen leicht den Überblick verlieren könnte. Ein größeres Problem sehe ich in der Einschränkung der Kommunikation zwischen den Gruppen. Die Vorgehensweise über Expertengruppen in der Designphase ist so schwierig umzusetzen, da für ähnliche Module verschiedene Anforderungen oder Schnittstellen benötigt werden. In sehr guten Kursen ist dieser Weg aber trotzdem gangbar. Man erhält so am Ende unterschiedliche Simulationen welche über die Anforderungsspezifikation und durch Tests sehr gut auf ihre Qualität geprüft werden können. Auch zur Leistungsbewertung ist dieser Weg geeignet.
Bei meinen Überlegungen steht aber der Entwicklungszyklus der objektorientierten Programmierung im Fordergrund. Deshalb entscheide ich mich dafür, dass die Schüler und Schülerinnen die vorgestellten Modelle durch Diskussion und Kompromissbildung zu einem Modell vereinigen. Was vermieden werden sollte, ist, ein Modell einer Gruppe allen anderen aufzuzwingen. Die einzelnen Modelle der Gruppen werden wahrscheinlich nicht extrem voneinander abweichen und man kann so aus den verschiedenen Überlegungen die besten herausfiltern und zu einem Modell zusammenführen. Dies bildet dann die Grundlage für die Designphase.

4.3.3. Verlaufsplanung

geschätzter Zeitaufwand: 1 Doppelstunde

Inhalt	Sozial- und Aktionsform	Schüleraktivität	Lehreraktivität
Objektkandidaten und Attribute finden Methoden bestimmen Beziehungen bestimmen	Gruppenarbeit	▫ lesen ▫ Textanalyse ▫ Abstrahieren ▫ diskutieren ▫ Verständnisprobleme klären ▫ UML-Diagramme erstellen	▫ Hilfestellung
Ergebnisvor-stellung	Schülervortrag Unterrichtsge-spräch	▫ einzelne Schüler stellen die Ergebnisse vor ▫ Ergebnisdiskussion zwischen den Gruppen ▫ entgültiges Modell finden	▫ wenn nötig Mode-rator

4.3.4. Ergebnis

Als Ergebnis dieser Phase steht ein Klassen- bzw. Objektmodell als Grundlage für die weiteren Phasen. (siehe Abb. 4.3.2.3.)

4.4. Designphase

4.4.1. Ziele

In dieser Phase

- können die Schüler die gefundene Klassenstruktur auf die Struktur eines lauffähigen Programms übertragen
- kennen die Möglichkeiten einer Programmiersprache in Bezug auf ihre objektorientierten Fähigkeiten
- üben sich im Modellieren
- können das Modulkonzept anwenden
- können das Programmieren im Großen (Architektur) und das Programmieren im Kleinen (Algorithmen innerhalb der Module) unterscheiden

4.4.2. methodisch-didaktische Überlegungen

In der Designphase kommen zur Übertragung der in der Analysephase gefundenen Struktur in eine Modulhierarchie Überlegungen hinzu, die sich aus der Realisierbarkeit und den Möglichkeiten der Programmiersprache ergeben. Weiterhin müssen Überlegungen zur Benutzeroberfläche und der Spezifizierung der Schnittstellen angestellt werden. Dabei müssen Entscheidungen über Datentypen, Speichermethoden oder Fehlerbehandlungen getroffen werden.
Bis zu diesem Zeitpunkt ist es völlig unerheblich, welche Programmiersprache man verwendet. Erst in der Designphase ist dies von belang. In meinem Beispiel verwende ich die

Programmiersprache Delphi. Jede andere Programmiersprache, welche eine objektorientierte Programmierung erlaubt, kann selbstverständlich auch genutzt werden.

Für die Benutzeroberfläche wird ein eigenes Objekt (Hauptformular) erstellt, welches die Schnittstelle zwischen dem Benutzer und der Registermaschine darstellt. Bei der Analyse der Benutzeroberfläche muss, wie oben beschrieben, ein Objektmodell erstellt werden. Diese Phase würde ich im Klassen- oder Kursverband im Unterrichtsgespräch durchführen um hier eine einheitliche Schnittstelle zur Registermaschine zu erhalten.
Ein Ergebnis könnte wie folgt aussehen:

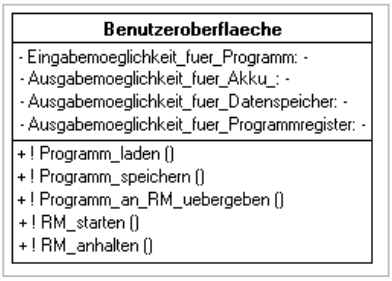


Abb.: 4.4.2.1. UML-Diagramm Benutzeroberfläche

Da die Implementation der Benutzeroberfläche erfahrungsgemäß sehr langwierig ist und über die Funktionalität und die Gestaltung sehr oft verschiedenste Meinungen aufeinanderprallen, kann man die Benutzeroberfläche schon als Modul vorgeben. Dies ist besonders in leistungsschwächeren Kursen empfehlenswert, da man die Schüler und Schülerinnen bei der Implementation entlastet und ihnen die notwendigen Funktionalitäten vorgibt.

Sind die Funktionalitäten der Benutzeroberfläche im Klassenverband geklärt, werden in den Gruppen Verantwortliche für die erstellten Module bestimmt. Je nach Gruppengröße können auch zwei Module einem Verantwortlichen übertragen werden (z.B.: den Daten und den Programmspeicher). Diese Verantwortlichen bilden jetzt für jedes Modul eine Expertengruppe in der die Designphase durchgeführt wird. Dies trägt zur Kommunikation mit Schülern und Schülerinnen anderer Gruppen bei, die vielleicht in ihrer Gruppe andere Sichtweisen auf die Module geprägt haben. Hierbei werden Verantwortlichkeit für die eigene Gruppe gefordert, die Diskussionsbereitschaft und die Teamfähigkeit gefördert.

In den Gruppen entstehen in der Designphase die Umsetzung des Modulkonzeptes in speziellen Units, bei der den Attributen spezielle Datenstrukturen und den Funktionalitäten spezielle Methoden oder Funktionen zugeordnet werden. Diese Methoden werden aber noch nicht ausprogrammiert. Des weiteren werden die Schnittstellen zwischen den Objekten definiert.
Ich kann an dieser Stelle nicht alle Überlegungen zu den Datentypen bzw. den Prozeduren und Funktionen für die einzelnen Module darstellen. Man kann diese an Hand der Unit-Entwürfe in 4.4.4. leicht nachvollziehen. Ich werde hier nur kurz auf zwei Aspekte, die zu beachten sind, eingehen.

Eine Regel bei der Modellierung sollte immer das Geheimnisprinzip sein. Das heißt: Soll ein Objekt Informationen speichern, so erhält es dazu eine Operation. Die Attribute der Objekte sind in der Regel nach außen versteckt (*private*) um Seiteneffekte zu vermeiden.

Bei der Modellierung der Assoziationen zwischen den Klassen bestehen zwei Möglichkeiten. Zum einen das Empfänger – Sender – Prinzip, in dem das, die Nachricht sendende Objekte sich selbst als Parameter übergibt. Somit kennt der Empfänger den Sender und kann

eventuelle Antworten richtig zuordnen. Eine zweite Möglichkeit ist bei der Initialisierung der Objekte einen festen Link als Objektattribut zu setzen. Ich werde beide Möglichkeiten exemplarisch darstellen.

Während der Designphase ist es notwendig, dass zwischen den Expertengruppen die Schnittstellen klar definiert sind und mögliche Veränderungen oder Spezifikationen über die Schnittstellenbeauftragten weitergeleitet werden.

Ist die Designphase abgeschlossen, kehren die Experten in ihre Gruppen zurück und stellen ihr entwickeltes Modul vor. Dabei werden noch notwendige Verständnisfragen geklärt.

4.4.3. Verlaufsplanung

geschätzter Zeitaufwand: 2-3 Stunde

Inhalt	Sozial- und Aktionsform	Schüleraktivität	Lehreraktivität
Benutzeroberfläche modellieren	Unterrichtsgespräch	▫ Abstrahieren ▫ diskutieren ▫ entwickeln Objektstruktur	▫ Hilfestellung
Module modellieren		▫ entwickeln einen Unitentwurf ▫ definieren Schnittstellen ▫ legen Datentypen und Prozeduren fest	
Module in den Gruppen zusammenführen	Schülervortrag Gruppengespräch	▫ Experten stellen die Modulentwürfe vor Schnittstellenkontrolle	

4.4.4. Ergebnis

Modulentwurf:

```
unit URegMasch;

interface

uses UTypen,UProzessor,UDatSpeicher,UProgSpeicher;

type
  TRegMasch = class
  private
     // Attribute aus der Analyse
     Datenspeicher : TDatenspeicher;
     Programmspeicher : TProgrammspeicher;
     Prozessor : TProzessor;
     // Objektverbindungen zur Benutzeroberfläche
     EingabeLink : TRichEdit;
     DatAusgabeLink : TStringGrid;
     AkkuAusgabeLink : TEdit;
     BefehlsAusgabeLink : TLabel;
     // Hilfsmethoden
     procedure Status_ausgeben;
     procedure Syntaxtest;
  public
     constructor create; virtual;
     destructor destroy; override;
     procedure reset;
     procedure init
     procedure Programmschritt_ausfuehren;
     procedure Programm_einlesen;
  end;

implementation

constructor TRegMasch.create;
{++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 +             Erzeugung des RM-Objekts                     +
 ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++}

destructor TRegMasch.destroy;
{++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 +            Löschen des RM-Objekts                        +
 ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++}

procedure TRegMasch.Status_ausgeben;
{++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 +       Übergibt den aktuellen Status der RM an die       +
 +         den Ausgabelink der Benutzeroberfläche          +
 ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++}

procedure TRegMasch.Syntaxtest;
{++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 + überprüft die Syntaktische Korrektheit der übergebenen  +
 + Programmzeile und gibt den entsprechenden Befehl zurück +
 ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++}

procedure TRegMasch.Programmschritt_ausfuehren;
{++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 +       RM führt einen Befehlszyklus aus                  +
 ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++}

procedure TRegMasch.Init;
{++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 +              RM wird initialisiert                      +
 ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++}

procedure TRegMasch.Reset;
{++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 +              RM wird zurückgesetzt                      +
 ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++}

procedure TRegMasch.Programm_einlesen;
{++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 +       Programmspeicher wird aus dem Eingabelink der     +
 +  Benutzeroberfläche ausgelesen, nach einem Syntaxtest   +
 +            im Programmspeicher abgelegt                 +
 +     und der Prozessor in Bereitschaft gesetzt           +
 ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++}
```

```
unit UProgSpeicher;

interface

uses UTypen;

type
  TProgrammspeicher = class
  private
     Befehl  : Array[1..999] of TBefehl;
  public
     constructor create;
     procedure init;
     procedure Befehl_speichern(index:Integer;Bef:TBefehl);
     function Befehl_ausgeben(index:Integer):TBefehl;
  end;

implementation

constructor TProgrammspeicher.create;
{++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 +             Erzeugung des Programmspeichers             +
 ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++}

procedure TProgrammspeicher.init;
{++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 +          Programmspeicher wird initialisiert            +
 ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++}

procedure
TProgrammspeicher.Befehl_speichern(index:Integer;Bef:TBefehl
);
{++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 +              RM-Befehl wird gespeichert                 +
 ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++}

function
TProgrammspeicher.Befehl_ausgeben(Index:Integer):TBefehl;
{++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 +              RM-Befehl wird ausgegeben                  +
 ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++}
```

```
unit UDatSpeicher;

interface

uses UTypen;

type
  TDatenspeicher = class
  private
     Wert    : Array[1..999] of LongInt;
  public
     constructor create; virtual;
     procedure init;
     procedure Wert_speichern(index:Integer;Dat:LongInt);
     function Wert_ausgeben(Index:Integer):LongInt;
  end;

implementation

constructor TDatenspeicher.create;
{++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 +             Erzeugung des Datenspeichers               +
 ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++}

procedure TDatenspeicher.init;
{++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 +            Datenspeicher wird initialisiert            +
 ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++}

procedure
TDatenspeicher.Wert_speichern(index:Integer;dat:LongInt);
{++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 +                  Wert wird gespeichert                  +
 ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++}

function
TDatenspeicher.Wert_ausgeben(Index:Integer):LongInt;
{++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 +                  Wert wird ausgegeben                   +
 ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++}
```

```
unit UProzessor;

interface

uses UTypen,UDatSpeicher,UProgSpeicher;

type
  TProzessor = class
  private
    // Objektverbindungen
    Datenlink : TDatenspeicher;
    Programmlink : TProgrammspeicher;
    //Attribute
    BefehlsRegister : TBefehl;
    Akkumulator     : LongInt;
    Befehlszaehler  : Integer;
  public
    constructor Create;virtual;
    procedure Befehlszyklus_ausfuehren;
    procedure Datenlink_setzen(Link:TDatenspeicher);
    procedure Programmlink_setzen(Link:TProgrammspeicher);
    procedure reset;
    procedure bereit_setzen;
    function  Befzaehler_ausgeben:Integer;
    procedure Status_ausgeben(Var Akku,BefZaehler:Integer;
                              Var BefReg:TBefehl)
  private
    procedure Befehl_holen;
    procedure Befehl_ausfuehren;
  end;

implementation

constructor TProzessor.Create;
{++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 +                Erzeugung des Prozessors                  +
 ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++}
procedure TProzessor.Status_ausgeben(Var Akku,BefZaehler:Integer;
                                     Var BefReg:TBefehl);
{++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 +         aktueller Status wird ausgegeben                 +
 ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++}
function  TProzessor.Befzaehler_ausgeben:Integer;
{++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 +       aktueller Befehlszähler wird ausgegeben            +
 ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++}
procedure TProzessor.Datenlink_setzen(Link:TDatenspeicher);
{++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 +     Verbindung zum Datenspeicher wird gesetzt            +
 ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++}
procedure TProzessor.Programmlink_setzen(Link:TProgrammspeicher);
{++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 +     Verbindung zum Programmspeicher wird gesetzt         +
 ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++}
procedure TProzessor.reset;
{++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 +               Prozessor wird zurückgesetzt               +
 ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++}
procedure TProzessor.bereit_setzen;
{++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 +         Prozessor wird in Bereitschaft gesetzt           +
 ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++}
procedure TProzessor.Befehl_holen;
{++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 +    aktueller Befehl wird aus dem Programmspeicher        +
 +           in das Befehlsregister geladen                 +
 ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++}
procedure TProzessor.Befehl_ausfuehren;
{++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 +               aktueller Befehl wird ausgeführt           +
 +            Operand holen, Befehl decodieren,             +
 +         Operation ausführen, Befehlszähler setzen        +
 ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++}
procedure TProzessor.Befehlszyklus_ausfuehren;
{++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 +             aktueller Befehl wird ausgeführt             +
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++}
```

```
unit UHaupt;  //Benutzeroberfläche

interface

uses  URegMasch;

type
  TF_Haupt = class(TForm)
    RE_Editor: TRichEdit;  //Programmeingabe
    SG_Daten: TStringGrid; //Datenspeicherausgabe
    L_Befehl: TLabel;              //Befehlsausgabe
    E_Akku: TEdit;           //Akkuausgabe
    procedure Code_laden;
    procedure Code_Speichern;
    procedure Prog_uebergabe;
    procedure Prog_ausfuehren_voll;
    procedure Prog_ausfuehren_schrittweise;
    procedure Hilfe_anzeigen;
end;

var
  F_Haupt: TF_Haupt;
  RegMasch : TRegMasch;

implementation

procedure TF_Haupt.Code_laden;
{++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 +        läd Programmcod aus Textdatei (*.rmp) in          +
 +                   Editorfenster                          +
 ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++}

procedure TF_Haupt.Code_Speichern;
{++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 +    speichert Programmcod aus Editorfenster in            +
 +                  Textdatei (*.rmp)                       +
 ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++}

procedure TF_Haupt.Prog_ausfuehren_voll;
{++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 +     Registermaschine arbeitet übergebenes Programm       +
 +                    vollständig ab                        +
 ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++}

procedure TF_Haupt.Prog_ausfuehren_schrittweise;
{++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 +     Registermaschine arbeitet übergebenes Programm       +
 +                    schrittweise ab                       +
 ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++}

procedure TF_Haupt.Prog_uebergabe;
{++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 +    Programm aus dem Editorfenster wird an die            +
 +              Registermaschine übergeben                  +
 ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++}

procedure TF_Haupt.Hilfe_anzeigen;
{++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 +    Hilfestellung zur Benutzung wird angezeigt            +
 ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++}
```

4.5. Implementierungsphase

4.5.1. Ziele

In dieser Phase sollen die Schüler und Schülerinnen

- gegebene Funktionalitäten in Algorithmen übertragen können.
- ihre Programmierfähigkeiten weiter entwickeln.
- Programmcode geeignet dokumentieren können.

4.5.2. methodisch-didaktische Überlegungen

In dieser Phase sollen die Schüler und Schülerinnen weitgehend selbstständig ihre Module programmieren. Die Gruppen führen am Ende die Module zu einem lauffähigen Programm zusammen. Dazu ist es notwendig, dass sich alle Schüler und Schülerinnen während der Implementierungsphase genau an die Schnittstellendefinitionen halten. Wichtig ist auch die Schüler und Schülerinnen zur Dokumentation des Quellcodes anzuhalten. Es müssen nicht alle Verzweigungen und Schleifen erläutert werden, aber man sollte eine Funktionsbeschreibung der einzelnen Prozeduren und Funktionen fordern.

Die Schüler müssen sich spätestens in dieser Phase, wenn dies nicht schon in der Designphase geschehen ist, für explizite Datentypen entscheiden. Diese Entscheidungen können sehr unterschiedlich ausfallen. Wichtig ist, dass die Schüler und Schülerinnen diese begründen können.
In der angegebenen Implementierung (siehe Anhang) wurden folgende Entscheidungen getroffen.

Benutzeroberfläche:
Für die Möglichkeit der Programmeingabe (Editor) bietet sich der Objekttyp des **TRichEdit** an. Es bietet Speicher- und Lademethoden (LoadFromFile, SaveToFile) und Editiermöglichkeiten wie Drag and Drop. Für die verschiedenen Ausgaben verwendet man am besten **Label**- oder **Edit**-Komponenten (für Akkumulator, Befehlszähler und den aktuellen Befehl) oder **StringGrid** für die Ausgabe des Datenspeichers. Alle weiteren Komponenten der Benutzeroberfläche dienen nur der Verbesserung der Bedienbarkeit.

Registermaschine:
Bei der Registermaschine sind die Hauptattribute Objekte der definierten Klassen **TDatenspeicher**, **TProgrammspeicher**, **TProzessor**. Die Registermaschine sowie die meisten anderen Klassen müssen mit Befehlen operieren. Dazu wurde ein Datentyp **TBefehl** in der Unit UTypen.pas definiert, der aus der Operation und dem Operanden besteht und allen Modulen zur Verfügung steht.

```
unit UTypen;
interface
type
 TOperation = String[6];
 TBefehl    = record
     Operation : TOperation;
     Operand   : Integer;
 end;
implementation
end.
```

Code Unit UTypen

Da die Registermaschine nur natürliche Zahlen verarbeitet, hätte als Datentyp für die Adressen, Daten bzw. den Befehlszähler der Typ Word oder LongWord ausgereicht. Es wurde aber **Integer** bzw. **LongInt** gewählt um Nichtbelegungen mit –1 oder das

Programmende mit dem Befehlszäher –2 markieren zu können. Diese Werte sind keine natürlichen Zahlen, können also nicht als reguläre Einträge der Registermaschine entstehen und sind so zur Codierung benötigter Zustände geeignet.
Daten- und Programmspeicher:
Der Datenspeicher wird in unserem Fall als **Array[1..999] of LongInt**, der Programmspeicher als **Array[1..999] of TBefehl** implementiert. Dies schafft für schulische Ansprüche genügend Speicherplatz und ist sehr leicht zu handhaben. Man könnte auch **dynamische Arrays** oder **Listendatentypen** erstellen. Dies würde das Niveau des Entwurfs und der Implementierung erhöhen und wäre somit eine gute Erweiterungsmöglichkeit.
Prozessor:
Beim Prozessor können der Akkumulator als **LongInt**, der Befehlszähler als **Integer** und das Befehlsregister als **TBefehl** implementiert werden. Die ALU aus der Analysephase, als ein physisches Schaltwerk, wird vom Computer ersetzt und entfällt. Dadurch werden auch einige Schritte des Befehlszykluses bei der Implementation zusammengefasst.

Die Umsetzung der Methoden durch geeignete Algorithmen hängt von den Programmierfähigkeiten der Schüler und Schülerinnen ab.

Zur Implementierungsphase muss auch eine Testphase gehören. Dazu sollte nach der Fertigstellung des Programms jede Gruppe die Registermaschinensimulation an vorgegebenen und selbsterstellten RM-Programmen testen.
Von Vorteil ist auch ein anschließender gegenseitiger Test der Programme durch die Gruppen. Dabei kann nicht nur die Funktionstüchtigkeit, sondern auch die Qualität der Benutzeroberfläche getestet und beurteilt werden. Die Gruppen können hierbei ihre Entscheidungen zur Funktionalität und zur Benutzeroberfläche begründen, verteidigen und gegebenenfalls korrigieren.

4.5.3. Verlaufsplanung

geschätzter Zeitaufwand: 2-3 Doppelstunden + Heimarbeit

Inhalt	Sozial- und Aktionsform	Schüleraktivität	Lehreraktivität
Implementation der Module	selbstständige Schülerarbeit	▫ entwerfen Algorithmen ▫ entwerfen ggf. Datentypen ▫ setzen dies in einer Programmiersprache um	▫ Hilfestellung
Module zusammenführen	Gruppenarbeit	▫ setzen die einzelnen Module im Baukastenprinzip zusammen ▫ testen ▫ korrigieren	
Programme testen	Klassen- / Kursverband	▫ testen ▫ diskutieren ▫ korrigieren ▫ bewerten	

4.5.4. Ergebnis

Programmcode, Programm (siehe Anhang) und Dokumentation.

5. Probleme und Alternativen

Für dieses Projekt ist es sehr schwer die benötigte Arbeitszeit abzuschätzen. Sie hängt sehr stark von den Fähigkeiten der Schüler und Schülerinnen und ihren Erfahrungen bei der Analyse, dem Design und dem Umgang mit UML und der verwendeten Programmiersprache ab. Gute Kurse können die vorgegebenen Zeiten bestimmt einhalten, bei schwächeren Kursen könnte dies zu Problemen führen. Besonders beim Vergleich und der Diskussion der Ergebnisse benötigen schwächere Gruppen mehr Zeit. Neben den fachlichen Qualifikationen sind dabei auch die Fähigkeiten bei der Präsentation, der Diskussion und der Reflexion zu berücksichtigen. Sollte es sich herausstellen, das die Schüler und Schülerinnen mehr Zeit als möglich benötigen sollte man Teile der Arbeit als Hausaufgabe erledigen lassen. Dies wäre aber kontraproduktiv zum Sinn der Gruppenarbeit, kann also nur während der Implementierungsphase eingesetzt werden.
Eine bessere Möglichkeit besteht darin, bestimmte Teilaspekte wie den Unitentwurf eines Moduls als Anschauungsbeispiel bereitzustellen oder bei der Implementation bestimmte Prozeduren, Typdeklarationen oder einzelne Units (z.B. Benutzeroberfläche) vorzugeben. Da der Entwicklungszyklus bei der objektorientierte Programmentwicklung im Mittelpunkt steht, kann man über diese Methoden, vor allem während der Implementation, Zeit sparen. Man sollte genügend Zeit für die Analyse und die Designphase investieren und lieber bei der Implementation sparen. Diskutiert werden in der letzten Zeit Ideen, ob die Implementation in der Schule noch diesen wichtigen Stellenwert bei der Softwareentwicklung haben sollte. Von guten Case-Tools wird ja heute schon ein großer Teil der Implementation übernommen. Meiner Meinung nach sollten alle Phasen der Softwareentwicklung gleichberechtigt behandelt werden. Die Entwicklung von geeigneten Algorithmen als Teil der Problemlösung ist genauso wichtig wie die Fähigkeiten bei der Abstraktion und Modellierung.
Ein weiteres Problem kann durch das unterschiedlich schnelle Arbeiten der Gruppen besonders der Expertengruppen auftreten. Die Modellierung und Implementierung des Programmspeichers bzw. des Datenspeichers ist weniger komplex als die der weiteren Module. Man kann die Speicher zu einem Modul zusammenfassen um den Arbeitsumfang anzugleichen oder die Ansprüche an die Speicher zu erweitern und wie oben beschrieben dynamische Arrays oder Listen zu verwenden. Dazu könnten neue Datenobjekte erstellt werden. Eine Möglichkeit wäre auch, dass sich leistungsschwächere Schüler und Schülerinnen mit diesen überschaubaren Modulen auseinandersetzen, Dies würde ihnen ein besseres Verständnis als bei komplexeren Modulen bieten. Sie wären aber nicht „ausgeschlossen“, da sie ein entscheidendes Modul für das spätere Programm schaffen, in ihren Gruppen vorstellen und verteidigen müssen. Dies könnte auch gruppendynamische Probleme in sehr heterogenen Gruppen entschärfen.
In der dargestellten Registermaschinensimulation gibt es noch einige nicht bearbeitete Probleme, die für die Schüler und Schülerinnen als Möglichkeit zur Weiterentwicklung des Programms dienen können. Es wurden z.B. keine Routinen zur Fehlerbehandlung (Überlaufs- oder Bereichsprüfung) implementiert. Des weiteren könnte man den Funktionsumfang oder die Benutzeroberfläche ändern. Die Ausgabe der einzelnen Arbeitsschritte der Registermaschine könnte z.B. in einer Programmablauftabelle erfolgen, die die Schüler und Schülerinnen schon aus der Erarbeitungsphase kennen. Man könnte auch eine Simulation entwerfen, in der der Benutzer zwischen verschiedenen Befehlssätzen wählen kann. Dies wäre für den Einsatz der Simulation in 13/I von Vorteil.
Alles in allem sollt man aber darauf achten, die Schüler und Schülerinnen nicht zu sehr in die vom Lehrer vorgedachten Bahnen zu lenken, sondern ihnen, wie es die Projektmethode fordert, viel Freiraum für Kreativität und ihre eigenen Vorstellungen zu lassen. Dies fordert vom Lehrer „lenken“ oder „laufen lassen“ im richtigen Moment, viel Überblick und auch den Mut zum eventuell nicht ganz ausgereiften Projektergebnis.

6. Literaturverzeichnis

[Ast/Bai] Asteroth / Baier: Theoretische Informatik
Pearson Studium 2002

[Bau1] Rüdeger Baumann: Informatik für die Sekundarstufe II Band 1
Klett 1993

[Bau2] Rüdeger Baumann: Informatik für die Sekundarstufe II Band 2
Klett 1993

[Erler] Dr. Thomas Erler: UML Das Einsteigerseminar
bhv 2000

[Ho/Ker/Fo] Horn / Kerner /Forbig: Lehr- und Übungsbuch Informatik Band 1
Fachbuchverlag Leipzig 2001

[Lehrplan] Lehrplan Informatik - Gymnasialer Bildungsgang –
Hessisches Kultusministerium 2003

[Louis] Dirk Louis: Delphi 5 , Markt+Technik 2000
Spektrum 2004

[Mod1] Eckart Modrow: Informatik mit Delphi Band 2
Dümmler 2000

[Mod2] Eckart Modrow: Virtuelle Lehrerweiterbildung Informatik in Niedersachsen

[Röhner] G. Röhner: Projektarbeit , Handreichung WBK VIII

[Schu/Schw] Sigrid Schubert / Andreas Schwill: Didaktik der Informatik
Spektrum Verlag 2004

[Six/Winter] Hans-Werner Six, Mario Winter: Software Engineering I
Vorlesungsscript 1793 Fernuni Hagen

[Spolwig] Siegfried Spolwig: Objektorientierung im Informatikunterricht
Dümmler 1997

7. Anhang

- Aufgabenstellung
- Material zur Registermaschine
- Arbeitsblatt
- Musterlösung
- Anforderungsspezifikation
- Quellcode

Programmierung einer Registermaschine unter objektorientierten Gesichtspunkten

1. Machen Sie sich an Hand der gegebenen Materialien mit dem Aufbau und der Funktionsweise der Registermaschine vertraut.

2. Entwickeln Sie eine Simulation der Registermaschine unter Berücksichtigung der objektorientierten Methoden.
   a) Analyse
   - Erstellen Sie eine Anforderungsspezifikation.
   - Entwickeln Sie einen Entwurf der benötigten Klassen und Objekte und ihren Beziehungen.

   b) Design
   - Entwickeln Sie eine geeignete Benutzeroberfläche.
   - Spezifizieren Sie das Klassenmodell in Bezug auf verwendete Datentypen und die erstellte Benutzeroberfläche.

   c) Implementierung
   - Programmieren Sie in einer objektorientierten Sprache die Registermaschine und testen Sie das Programm mit Hilfe gegebener und selbstentwickelter Registermaschinenprogramme.

3. Erstellen Sie eine Dokumentation über die einzelnen Phasen der Projektarbeit.

Die Registermaschine

Die Registermaschine ist ein programmgesteuerter Automat mit zugehörigen Programmierbefehlen. Sie modelliert sehr stark vereinfacht das Wesen der Programmsteuerung von Mikroprozessoren.

Aufbau:

Die Registermaschine besteht aus:

(1) Speicher

Der **Speicher** ist eine lineare Anordnung gleich großer Speicherzellen, deren jede eine Zahl oder Zeichenkette aufnehmen kann und eine Nummer, die sogenannte Adresse besitzt. Wir unterteilen den Speicher in Programmspeicher und Datenspeicher. Im **Programmspeicher** sind die von der Maschine auszuführenden Befehle enthalten. Der **Datenspeicher** nimmt die zu verarbeitenden Daten auf und bietet Platz für eventuelle Zwischenergebnisse.

(2) Prozessor

Der **Prozessor** besteht aus Rechenwerk und Steuerwerk. Letzteres, auch Leitwerk genannt, organisiert die Programmausführung. Das Rechenwerk, die arithmetisch-logische Einheit(ALU) ist ein Schaltwerk, welches auf Veranlassung des Steuerwerks sämtliche arithmetischen und logischen Operationen durchführt. Wichtiger Bestandteil des Prozessors sind die folgenden Speicherbausteine (Arbeitsregister).

- Das erste Arbeitsregister ist der **Akkumulator**: Er nimmt jeweils den Inhalt einer Zelle des Datenspeichers auf; den Namen (lat.: accumulare = sammeln) hat er von seiner Funktion, die Rechenergebnisse zu sammeln. das heißt: vor Ausführung der zweistelligen Operation befindet sich der eine Operand im Datenspeicher, der andere im Akkumulator; nach der Ausführung befindet sich das Ergebnis im Akkumulator.
- Das zweite Arbeitsregister ist das **Befehlsregister** : Es nimmt jeweils den Inhalt einer Zelle des Programmspeichers, und zwar den momentan auszuführenden Befehl, auf.
- Das dritte Arbeitsregister ist das Befehlsadressregister (kurz **Befehlszähler**). Es beinhaltet jeweils eine Adresse des Programmspeichers, nämlich die des Befehls, der gerade zur Ausführung ansteht.

Befehlssatz:

C[i] - *Inhalt der Datenspeicherzelle mit der Adresse i ;* *C[0]* - *Inhalt des Akkumulators ,*
b - *Befehlszähler(Transportbefehle)*

load i - C[0]=C[i]; b:=b+1;
cload i - C[0]=i; b:=b+1;
iload i - C[0]=C[C[i]]; b:=b+1;
store i - b = b + 1; C[i] = C[0];
istore i - if (c(i)>=1) then {b = b + 1};
C[C[i]] = C[0]; } else {b = b};

(Sprungbefehle)

goto i - b = i;
igoto i - b = C[i];
jzero i - if (C[0]==0) b = i; else b = b+1;

(Verarbeitungsbefehle)

add i - b = b + 1; C[0] = C[0] + C[i];
cadd i - b = b + 1; C[0] = C[0] + i;
iadd i - b = b + 1; C[0] = C[0] + C[C[i]];
sub i - b = b + 1; if (C[0]>=C[i]) then
{C[0]=C[0] - C[i];} else {C[0]=0;}
csub i - b = b + 1; if (C[0]>=C[C[i]]) then
{C[0]=C[0] - C[C[i]];} else {C[0]=0;}
isub i - b = b + 1; if (C[0]>=i) then
{C[0] = C[0] - i;} else {C[0] = 0;}
mult i - b = b + 1; C[0] = C[0] * C[i];
cmult i - b = b + 1; C[0] = C[0] * i;
imult i - b = b + 1; C[0] = C[0] * C[C[i]];
div i - if (C[i]>0) then
{C[0]=C[0] / C[i]; b=b+1;} else (b=b);
cdiv i - if (i>0) then
{C[0]=C[0] / i; b=b + 1;} else (b = b);
idiv i - if (C[C[i]]>0) then
{C[0] = C[0] / C[C[i]]; b = b + 1;} else (b = b);

(Terminierung)

end - Programmende

Man unterscheidet bei den Befehlen zwischen Operationen mit direkter Adressierung (add i, load i), mit indirekter Adressierung (iadd i, iload i, ...) oder konstanten Operanden (cadd i, cload i, ...). Bei der direkten Adressierung wird auf den Inhalt des Registers C[i], bei der indirekten Adressierung auf den Inhalt des Registers C[C[i]] dessen Adresse in Zelle C[i] steht.

Funktionsweise:

Zur Abarbeitung der Befehle durchläuft die Registermaschine einen fünfschrittigen Befehlszyklus:

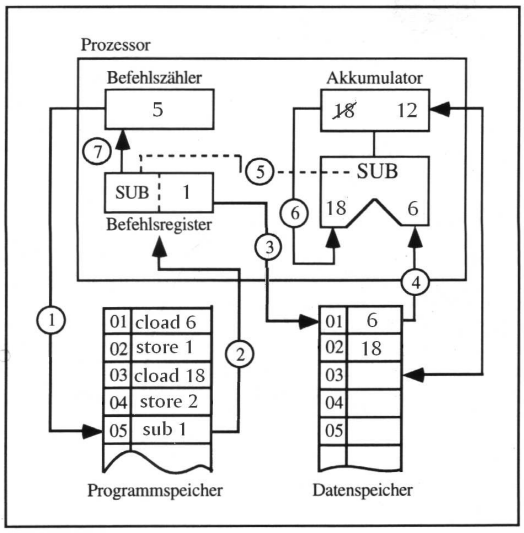
6. Befehl holen (1,2)
Aus dem Programmspeicher wird das Befehlswort (Befehl und Operand) in das Befehlsregister geladen, welches die Adresse des Wertes im Befehlszählers besitzt.

7. Operand holen (3,4)
Ist der Operand des Befehlswortes im Befehlsregister eine Adresse, so wird mit dieser Adresse der Operand für die Operation aus dem Datenspeicher in die ALU geladen. Ansonsten wird der Operand des Befehlswortes in die ALU geladen.

8. Befehl dekodieren (5)
Der ALU wird mitgeteilt, welche Operation ausgeführt werden soll.

9. Operation ausführen (6)
Die ALU holt den Wert aus dem Akkumulator und führt die Operation durch. Das Ergebnis steht anschließend im Akkumulator.

10. Befehlszähler ändern (7)
Nach der Durchführung des Befehls wird der Befehlszähler in der Regel um den Wert 1 erhöht. Ausnahmen bilden die Sprungbefehle. In diesen Fällen wird die Sprungadresse des Befehls an den Befehlszähler übergeben.



Befehl holen (1, 2), Operand holen (3, 4), Befehl dekodieren (5), Operation ausführen (6), Befehlszähler ändern (7)

Beispiel Maximumbestimmung (max(6, 18))

Programm

	Befehl
1	cload 6
2	store 1
3	cload 18
4	store 2
5	sub 1
6	jzero 9
7	load 2
8	goto 10
9	load 1
10	end

Programmablauf:

Befehls-adresse	***Befehl***	***Bedeutung***	***C[0] Akku***	***C[1]***	***C[2]***	***Bef.zähler neu***
1	cload 6	lädt 6 in Akku	6			2
2	store 1	speichert Akku in Register C[1]	6	6		3
3	cload 18	lädt 18 in Akku	18	6		4
4	store 2	speichert Akku in Register C[2]	18	6	18	5
5	sub 1	subtrahiert von Akku den Inhalt aus Register C[1], Ergebnis steht im Akku wenn Ergebnis > 0 ansonsten Akku = 0	12	6	18	6
6	jzero 9	wenn Akku == 0 dann zu Befehlszeile 9 ansonsten b = b+1	12	6	18	7
7	load 2	lädt den Inhalt aus Register C[2] in den Akku	18	6	18	8
8	goto 10	Sprung zu Befehlsadresse 10	18	6	18	10
10	end	Programmende, Ergebnis steht im Akku	**18**	6	18	

1. Vervollständigen Sie die Programmablauftabelle für das angegebene Programm!

RM-Programm:

	Befehl
1	cload 4
2	store 1
3	csub 1
4	jzero 10
5	store 2
6	mult 1
7	store 1
8	load 2
9	goto 3
10	load 1
11	end

Befehls-adresse	***Befehl***	***Bedeutung***	***C[0] Akku***	***C[1]***	***C[2]***	***Bef.zähler neu***
1	cload 4	lädt 4 in Akku	4			2

2. Erstellen Sie ein Registermaschinenprogramm zur Berechnung von n^m am Beispiel 4^3 .

1. Programmablauftabelle

Befehls-adresse	***Befehl***	***Bedeutung***	***C[0] Akku***	***C[1]***	***C[2]***	***Bef.zähler neu***
1	cload 4	lädt 4 in Akku	4			2
2	store 1	speichert Akku in C[1]	4	4		3
3	csub 1	subtrahiert vom Akku 1	3	4		4
4	jzero 10	wenn Akku = 0 dann b = 10 ansonsten b = 4+1	3	4		5
5	store 2	speichert Akku in C[2]	3	4	3	6
6	mult 1	multipliziert Akku mit C[1]	12	4	3	7
7	store 1	speichert Akku in C[1]	12	12	3	8
8	load 2	lädt C[2] in Akku	3	12	3	9
9	goto 3	setze Bef.zähler auf b = 3	3	12	3	3
3	csub 1	subtrahiert vom Akku 1	2	12	3	4
4	jzero 10	wenn Akku = 0 dann b = 10 ansonsten b = 4+1	2	12	3	5
5	store 2	speichert Akku in C[2]	2	12	2	6
6	mult 1	multipliziert Akku mit C[1]	24	12	2	7
7	store 1	speichert Akku in C[1]	24	24	2	8
8	load 2	lädt C[2] in Akku	2	24	2	9
9	goto 3	setze Bef.zähler auf b = 3	2	24	2	3
3	csub 1	subtrahiert vom Akku 1	1	24	2	4
4	jzero 10	wenn Akku = 0 dann b = 10 ansonsten b = 4+1	1	24	2	5
5	store 2	speichert Akku in C[2]	1	24	1	6
6	mult 1	multipliziert Akku mit C[1]	24	24	1	7
7	store 1	speichert Akku in C[1]	24	24	1	8
8	load 2	lädt C[2] in Akku	1	24	1	9
9	goto 3	setze Bef.zähler auf b = 3	1	24	1	3
3	csub 1	subtrahiert vom Akku 1	0	24	1	4
4	jzero 10	wenn Akku = 0 dann b = 10 ansonsten b = 4+1	0	24	1	10
10	load 1	lädt C[1] in den Akku	24	24	1	11
11	end	Ende, im Akku steht Ergebnis	**24**	24	1	

2. Programm zur Berechnung n^m am Beispiel 4^3 .

Übergabe der Ausgangsdaten

	Befehl
1	cload 4
2	store 1
3	cload 3
4	store 2
5	cload 1
6	store 3

Schleife

	Befehl
7	load 2
8	jzero 15
9	csub 1
10	store 2
11	load 3
12	mult 1
13	store 3
14	goto 7

Ausgabe

	Befehl
12	load 3
13	end

Anforderungsdefinition zur Simulation einer Registermaschine

Die Registermaschine stellt ein einfaches Rechnermodell dar.

Es besteht aus:

- einem Datenspeicher, dessen adressierbare Zellen natürliche Zahlen speichern können.
- einem Programmspeicher, dessen adressierbare Zellen RM-Befehle speichern können.
- einem Prozessor. Der besitzt:
  - einem Akkumulator, der
  - ein Befehlsregister, der einen RM-Befehl speichern kann.
  - einen Befehlszähler, der eine natürliche Zahl speichern kann.
  - einer ALU welche die Operationen durchführt.

Grundfunktion
Die Registermaschine muss ein RM-Programm im Programmspeicher ablegen, dieses über den Befehlszyklus abarbeiten können. Das (korrekte) Endergebnis steht am Ende im Akkumulator.

Des weiteren muss die Simulation eine Schnittstelle für den Benutzer bieten, über die ein RM-Programm übergeben werden und das Ergebnis ausgelesen werden kann.

Funktionserweiterung
Auf der Benutzeroberfläche soll eine Darstellung der aktuellen Belegungen des Datenspeichers, des Akkumulators und des Befehlsregisters möglich sein.

Versicherung

Ich versichere, diese Arbeit selbständig verfasst, keine anderen als die angegebenen Hilfsmittel verwendet und die Stellen, die in anderen Werken im Wortlaut, als Graphik oder dem Sinne nach entnommen sind, mit Quellenangaben kenntlich gemacht zu haben.

Hofheim, den 05.09.2004