

Christian Maurer

Mini

v. 17. September 2011

Dr. Christian Maurer
Keithstr. 16
10787 Berlin

<http://murus.org/>

Die Quelltexte des des Modellen ^{Ula} Univer ^{Sums} sind mit größter Sorgfalt entwickelt und werden laufend gepflegt. Kein Programmsystem dürfte jedoch jemals frei von Fehlern sein; deshalb ist auch nicht damit zu rechnen, daß *dieses* System fehlerfrei ist: Es darf nur benutzt werden „wie es ist“.

Die Modula-2- und die Java-Quelltexte von Murus sind freie Software. Sie können sie unter den Bedingungen der GNU General Public License, wie von der Free Software Foundation veröffentlicht, weitergeben und/oder modifizieren, entweder gemäß Version 3 der Lizenz oder (nach Ihrer Wahl) jeder späteren Version. Ihre Veröffentlichung erfolgt in der Hoffnung, daß sie Ihnen von Nutzen sein könnten – aber *ohne irgendeine Garantie*, auch ohne die implizite Garantie der *Marktreife* oder der *Verwendbarkeit für einen bestimmten Zweck*.

Der Originaltext der GPL ist im weltweiten Netz unter der Adresse www.gnu.org/licenses/gpl.html zu finden; eine deutsche Übersetzung unter www.gnu.de/documents/gpl-3.0-de.html.

Die Quelltexte von Murus sind nur zu Lehrzwecken konstruiert und haben rein akademischen Wert. Ihre Verwendung in Programmen könnte zu *Schäden* führen, z. B. zur Inbrandsetzung von Rechnern, zur Entgleisung von Eisenbahnzügen, zum GAU in Atomkraftwerken oder zum Absturz des Mondes ...

Satz: Autor mit $\mathcal{A}\mathcal{M}\mathcal{S}$ - $\text{T}\mathcal{E}\mathcal{X}$

Wenn ein Rechner nicht richtig programmiert ist,
ist damit zu rechnen, dass er nicht richtig rechnet.

Vorwort

Vor vielen Jahren wurde mir eine äußerst lehrreiche und wohlüberlegte Unterrichtsreihe zur Informatik von C. LAURETTI und H. GEBAUER am Goethe-Gymnasium in Berlin-Wilmersdorf bekannt, die mir sehr gut gefallen hatte.

Ihre Ideen und Konzepte waren der Anlaß für dieses Projekt.

Christian Maurer

Inhaltsverzeichnis

ALLGEMEINES

Arbeitsverzeichnis	1
--------------------------	---

MINIPROGRAMME

Das Konzept	2
Bestandteile von Mini	3
Prozessor	3
Datenspeicher	4
Programmzeilen	5
Ausführung eines Miniprogramms	6
Anweisungen	7
lda, sta, exa, ldb, stb, exb	8
ina, dea, inc, dec, shl	9
shr, add, adc, sub	10
mul, div, jmp, je	11
jne, jc, jnc, cmp, clc, stc	12
cmc, push, pop, call, ret	13
Beispiele	14

DAS PROGRAMM MINI

Das Simulationsprogramm Mini	15
Allgemeines zur Programmbedienung	16
Hinweise zur Arbeit mit Mini	17

ANHANG

Aufgaben	18
----------------	----

ALLGEMEINES

Mini ist als Bestandteil des $\text{Modul}_{\text{Univer}}^{\text{Ula}}\text{Sums}$ nach dessen Installation oder Aktualisierung ebenfalls installiert.

Voraussetzung für die Installation des $\text{Modul}_{\text{Univer}}^{\text{Ula}}\text{Sums}$ ist die Installation von **Mocka**, dem Modula-2-Compiler der GMD (Hinweise dazu finden sich unter lwb.mi.fu-berlin.de/inf/mocka).

Arbeitsverzeichnis

Das Arbeitsverzeichnis, aus dem heraus **Mini** aufgerufen wird und in dem ggf. Daten abgelegt werden, ist $\$HOME/.Mini$ (es sei denn, daß durch die Umgebungsvariable **Mini** ein anderes Arbeitsverzeichnis definiert ist). Damit ist sichergestellt, daß Benutzer/innen mit dem Aufruf von **Mini** ihre eigenen Daten verwalten.

Wenn dieses Verzeichnis noch nicht existiert, wird es beim ersten Aufruf von **Mini** selbsttätig angelegt und es werden Beispieldaten aus dem $\text{Modul}_{\text{Univer}}^{\text{Ula}}\text{Sum}$ dorthinein kopiert, falls es welche gibt.

MINIPROGRAMME

Das Konzept

Bei **Mini** handelt es sich um ein einfaches Modell einer Einadreß-Maschine mit einem begrenzten Satz von Anweisungen, das geeignet ist, Grundkonzepte der Maschinenprogrammierung propädeutisch zu behandeln.

Die Bedeutung dieses Konzepts liegt

- *theoretisch* in der Turing-Vollständigkeit von **Mini**, da es offenbar nicht weniger leistungsfähig ist als das der $R_{\text{Register}}^{\text{Maschine}}$:
 - *Alles, was sich überhaupt programmieren läßt, läßt sich daher im Prinzip schon mit **Mini** erledigen.*
- *unterrichtspraktisch* in seiner Klarheit und Verständlichkeit: Miniprogramme sind unkompliziert und leicht handhabbar; mit den vorhandenen Anweisungen – bei äußerst geringem syntaktischen Aufwand – aber doch a priori schon recht ausdrucksstark (der Satz der Anweisungen ist wegen der streng objektbasierten Architektur des Systems leicht erweiterbar);
- *methodisch* in seinem einführenden Charakter in grundlegende Konzepte der *maschinennahen* sowie allgemein der *imperativen* Programmierung: Der Zustandsbegriff (repräsentiert durch *Wertzuweisungen an Variable*) und die algorithmischen Grundstrukturen (*Sequenz*, *Fallunterscheidung* und *Schleife*) lassen sich auf natürliche Weise mit sehr einfach zu erstellenden Miniprogrammen entwickeln.

Das Konzept stellt damit eine didaktisch-methodisch leicht zugängliche und in vieler Hinsicht vorteilhafte Variante zur Einführung in die imperative – insbesondere die maschinennahe – Programmierung dar.

Das Thema ist auch gut für eine Unterrichtsreihe in frühen Phasen des Informatikunterrichts geeignet, weil es keine Vorkenntnisse voraussetzt, insbesondere auch in einem Profilkurs.

Bestandteile von Mini

Mini verfügt über

- einen *Prozessor* zur Ausführung eines Programms,
- einen *Datenspeicher* in Form einer Menge von *Registern*,
- einen *Stapelspeicher* zur Zwischenspeicherung von Registerinhalten,
- einen *Programmspeicher* zur Aufnahme eines Programms in Form einzelner *Programmschritte*.

Der Datenspeicher ist begrenzt, er umfaßt 20 *Register* (Details siehe nächste Seite). Die Größe des Stapel- und des Programmspeichers ist nicht festgelegt; es wird aber vorausgesetzt, daß beide groß genug sind.

Darüberhinaus benötigt **Mini** eine Möglichkeit

- zur Eingabe eines Programms, d. h. zur Belegung des Programmspeichers mit Programmschritten sowie
- zur Belegung der Register mit Daten und zur Ausgabe dieser Daten.

Prozessor

Minis *Prozessor* hat die Aufgabe,

- mittels eines *Rechenwerks* die einzelnen Programmschritte und
- mittels eines *Steuerwerks* das Programm, d. h. die Folge der Programmschritte, auszuführen (Details siehe S. 11).

Er verfügt dazu über eigene *Register*:

- zwei *Akkumulatorregister* **ax** und **bx** (kurz als *Akkus* bezeichnet),
- einen *Programmzähler* zur Steuerung der Programmausführung sowie
- ein *Statusregister*, in dem einige Anweisungen bestimmte *Flaggen*
 - die *Nullflagge* **zf** und
 - die *Ausleihflagge* (beim Addieren/Subtrahieren „eins im Sinn“ = „eins geborgt“) **cf** setzen oder löschen.

Zu Beginn der Ausführung eines Miniprogramms hat die Nullflagge **zf** den Wert 1, alle anderen Register haben den Wert 0.

Datenspeicher

Der Datenspeicher von **Mini** besteht aus 20 *Registern* und dem *Stapelspeicher*.

Register sind Speicherplätze, die jeweils eine (maximal 9-stellige) natürliche Zahl aufnehmen können. Die Inhalte der Register werden auch als ihr *Wert* bezeichnet.

Da die Werte der Register im Laufe der Ausführung eines Programms verändert werden (was ja gerade der Zweck von Programmen ist), können die Register auch als *Variable* betrachtet werden. Alle Register haben einen Namen, über den sie angesprochen werden, um an ihre Werte heranzukommen.

Der *Stapelspeicher* besteht – anschaulich gesprochen – aus „übereinanderliegenden“ Registern (nach dem „*last in-first out*“-Prinzip); Registerwerte können jeweils oben auf dem Stapel abgelegt oder von oben entnommen werden, wobei darauf zu achten ist, daß ein Wert nur *dann* entnommen werden kann, wenn der Stapel nicht leer ist).

Zu Beginn der Ausführung eines Miniprogramms haben alle Register den Wert 0 und der Stapelspeicher ist leer.

Programmzeilen

Die einzelnen Schritte eines **Mini**-Maschinenprogramms (im folgenden kurz Miniprogramm genannt) stehen „Zeile für Zeile“ hintereinander im Programmspeicher und werden deshalb als *Programmzeilen* bezeichnet. Sie werden – beginnend bei 0 – fortlaufend durchnumeriert; in der Reihenfolge, wie sie im Programmspeicher stehen.

Jede Programmzeile umfaßt genau eine *Anweisung* (Details siehe S. 12), und zwar

- eine *Speicheranweisung*,
- eine Anweisung für eine *Rechenoperation*,
- eine *Vergleichsanweisung*,
- eine *Flaggenanweisung*,
- eine *Sprunganweisung*,
- eine *Stapelanweisung*,
- eine *Aufrufanweisung* oder
- eine *Rückkehranweisung*.

Am Anfang einer Programmzeile darf zusätzlich eine *Marke* stehen.

Ausführung eines Miniprogramms

Ein Miniprogramm wird von **Mini** in der Weise ausgeführt, daß seine Programmzeilen *sequentiell* (d. h. Zeile für Zeile) abgearbeitet werden, wobei bei der ersten Zeile begonnen wird und danach entweder zur nächsten Programmzeile gegangen oder zu einer anderen als der nächsten Zeile gesprungen wird.

Etwas genauer betrachtet, wird das so realisiert:

Als jeweils nächste Programmzeile wird immer *diejenige* Zeile ausgeführt, deren Nummer im *Programmzähler* des Prozessors steht.

Anfangs enthält der Programmzähler eine 0; es wird also bei der ersten Programmzeile begonnen. Welche Zahl *nach* der Ausführung einer Programmzeile im Programmzähler steht, d. h. welche Zeile als nächste ausgeführt wird, hängt davon ab, ob die Programmzeile eine Sprung- oder Aufrufanweisung enthält.

Ein Miniprogramm wird beendet, wenn es auf eine Zeile mit der Rückkehranweisung trifft; folglich ist bei der Erstellung eines Miniprogramms darauf zu achten, daß es eine Rückkehranweisung enthält.

Wenn der Wert des Programmzählers größer oder gleich der Anzahl der Programmschritte ist, wird das Programm abgebrochen, da die Numerierung mit 0 beginnt, z. B. nach der letzten Programmzeile, falls das keine Sprunganweisung ist.

Ein Programmabbruch kann auch andere Ursachen haben – und zwar „Programmierfehler“ in Form nicht beachteter Voraussetzungen beim Aufruf gewisser *Anweisungen*.

Ein *Unterprogramm* besteht aus einer Folge von Programmzeilen, deren *erste* mit einer Marke eingeleitet wird und deren *letzte* aus der Rückkehranweisung **ret** besteht. Es wird durch die dafür vorgesehene *Aufrufanweisung* ausgeführt, wobei in den Programmzähler bei Beendigung des Unterprogramms die Nummer der nächsten Programmzeile eingesetzt wird (die auf diejenige Zeile folgt, in der das Unterprogramm aufgerufen wurde).

Anweisungen

Mini verfügt nur über eine beschränkte „Programmiersprache“ mit wenigen *Anweisungen*:

- sechs *Speicheranweisungen* zum Kopieren von Registerinhalten in den/aus dem Akku **ax**: **lda**, **sta**, **exa**, **ldb**, **stb** und **exb**;
- vier Anweisungen zum *Erhöhen und Erniedrigen* der Werte von Akku und Registern: **ina**, **dea**, **inc** und **dec**;
- zwei *Schiebeanweisungen* zur Multiplikation bzw. Division von Registerwerten mit bzw. durch 2: **shl** und **shr**;
- fünf Anweisungen zur Ausführung von *Rechenoperationen* auf den Werten (temporär der Akkus und) der Register: **add**, **adc**, **sub**, **mul** und **div**;
- einer Anweisung zum *Vergleich* von Akku und Registerwerten: **cmp**;
- drei *Flaggenanweisungen* zur Manipulation des Statusregisters: **clc**, **stc** und **cmc**;
- fünf *Sprunganweisungen* zum „Springen“ im Programm, auch in Abhängigkeit von den Werten der Flaggen im Statusregister: **jmp**, **je**, **jne**, **jc** und **jnc**;
- zwei *Stapelanweisungen* zur Zwischenspeicherung von Werten von Registern: **push** und **pop**;
- einer *Aufrufanweisung* zur Ausführung eines Unterprogramms: **call** und
- einer *Rückkehranweisung* zur Beendigung eines Programms: **ret**.

Speicheranweisungen erwarten ein Register, *Sprunganweisungen* und die *Aufrufanweisung* eine Marke als Argument.

Anweisungen zum *Erhöhen* bzw. *Erniedrigen*, zur Ausführung von *Rechenoperationen* sowie *Schiebe-* und *Vergleichsanweisungen* erwarten höchstens *ein* und die *Stapelanweisungen* genau *ein* Argument:

Die Operanden einstelliger Operationen sind ein Akku *oder* ein als Argument übergebenes Register, zweistellige Operationen arbeiten auf einem Akku *und* einem als Argument übergebenen Register.

Alle anderen Anweisungen erwarten kein Argument.

Da die Programmzeilen nur aus Anweisungen mit höchstens *einem* Argument bestehen, ist **Mini** ein Beispiel für eine *Einadreßmaschine*.

Im Einzelnen sind die Anweisungen wie folgt spezifiziert (wobei die Flaggen nicht gesetzt oder gelöscht werden, wenn das nicht explizit angegeben ist):

`lda R`

Effekt:

Der Wert des Registers `R` ist in den Akku `ax` kopiert; danach steht die Nummer der nächsten Programmzeile im Programmzähler, was zur Folge hat, daß danach die nächste Programmzeile ausgeführt wird.

`sta R`

Effekt:

Der Wert des Akkus `ax` ist in das Register `R` kopiert; danach steht – wie oben – die Nummer der nächsten Programmzeile im Programmzähler.

`exa R`

Effekt:

Die Werte des Akkus `ax` und des Registers `R` sind vertauscht, d. h. beide Register enthalten jetzt den Wert, der vorher im jeweils anderen stand.

`ldb R`

Effekt:

Analog zu `lda R`, aber mit dem Akku `bx`.

`stb R`

Effekt:

Analog zu `sta R`, aber mit dem Akku `bx`.

`exb R`

Effekt:

Analog zu `exa R`, aber mit dem Akku `bx`.

ina

Effekt:

Wenn der Wert des Akkus **ax** kleiner als $10^9 - 1$ war, ist er um 1 erhöht, sonst auf 0 gesetzt; der Programmzähler ist – wie oben – verändert. Die Nullflagge ist gesetzt, wenn der Akku **ax** jetzt den Wert 0 hat, und gelöscht, wenn der Wert von **ax** nicht 0 ist.

dea

Effekt:

Wenn der Wert des Akkus **ax** größer als 0 war, ist er um 1 erniedrigt, sonst hat er jetzt den Wert $10^9 - 1$; der Programmzähler ist – wie oben – verändert. Die Nullflagge ist wie bei **ina** gesetzt bzw. gelöscht.

inc R

Effekt:

Wenn der Wert des Registers **R** kleiner als $10^9 - 1$ war, ist er um 1 erhöht, sonst auf 0 gesetzt; der Programmzähler ist – wie oben – verändert. Wenn das Register **R** jetzt den Wert 0 hat, ist die Nullflagge **zf** gesetzt, sonst gelöscht.

dec R

Effekt:

Wenn der Wert des Registers **R** größer als 0 war, ist er um 1 erniedrigt, sonst unverändert; der Programmzähler ist – wie oben – verändert. Die Nullflagge ist wie bei **inc R** gesetzt bzw. gelöscht.

shl R

Effekt:

Wenn das Doppelte des Wertes des Registers **R** kleiner als 10^9 war, ist er verdoppelt und die Ausleihflagge gelöscht.

Andernfalls enthält das Register **R** jetzt das Doppelte seines vorherigen Wertes $\text{mod } 10^9$ und die Ausleihflagge ist gesetzt.

Die Nullflagge ist wie bei **inc R** gesetzt bzw. gelöscht.

shr R

Effekt:

Das Register **R** enthält jetzt die Hälfte seines vorherigen Wertes; die Ausleihflagge ist gesetzt, wenn der vorherige Wert von **R** eine ungerade Zahl war, sonst gelöscht.

Die Nullflagge ist wie bei **inc R** gesetzt bzw. gelöscht.

add R

Effekt:

Der Wert des Akkus **ax** ist um den Wert des Registers **R** erhöht, sofern das ohne Überlauf über $10^9 - 1$ möglich ist; in diesem Fall ist die Ausleihflagge **cf** gelöscht.

Andernfalls enthält der Akku **ax** die Summe der Werte von **ax** und **R mod 10^9** und die Ausleihflagge ist gesetzt.

adc R

Effekt:

Der Wert des Akkus **ax** ist um den Wert des Registers **R** und zusätzlich um 1 erhöht, wenn die Ausleihflagge **cf** gesetzt war, sofern das ohne Überlauf über $10^9 - 1$ möglich ist. In diesem Fall ist die Ausleihflagge **cf** jetzt gelöscht.

Andernfalls enthält der Akku **ax** die Summe der Werte des Akkus **ax**, des Registers **R** und der Ausleihflagge **cf mod 10^9** und die Ausleihflagge ist jetzt gesetzt.

sub R

Effekt:

Wenn der Wert des Registers **R** nicht größer als der des Akkus **ax** ist, ist der Akku **ax** um diesen Wert vermindert und die Ausleihflagge ist gelöscht.

Andernfalls, d. h. wenn der Wert von **R** größer als der von **ax** ist, hat der Akku **ax** jetzt das Komplement der Differenz dieser Werte zu 10^9 , d. h. den Wert $10^9 - (\text{Wert von } R - \text{Wert von } ax)$ und die Ausleihflagge ist gesetzt. Die Nullflagge ist gesetzt, wenn **ax** jetzt den Wert 0 hat, und gelöscht, wenn der Wert von **ax** nicht 0 ist.

mul R

Effekt:

Der Wert des Akkus **ax** ist um den Wert des Registers **R** vervielfacht, sofern das ohne Überlauf über $10^9 - 1$ möglich ist; in diesem Fall ist die Ausleihflagge **cf** gelöscht.

Andernfalls enthält der Akku **ax** das Produkt der Werte von **ax** und **R** mod 10^9 und der Akku **bx** den Wert dieses Produktes div 10^9 (d. h. das Produkt ist die Zahl Wert von **bx** · 10^9 + Wert von **ax**); in diesem Fall ist die Ausleihflagge **cf** jetzt gesetzt.

div R

Effekt:

Wenn das Register **R** den Wert 0 enthält oder wenn der Wert des Akkus **bx** größer oder gleich dem des Registers **R** ist, wird das Programm abgebrochen.

Andernfalls ist die Summe aus dem Wert des Akkus **ax** und dem 10^9 -fachen des Wertes des Akkus **bx** durch den Wert von **R** geteilt; der Akku **ax** enthält jetzt den Quotienten (ohne Rest) und der Akku **bx** den Divisionsrest.

jmp M

Effekt:

Die Nummer der ersten Programmzeile mit der Marke **M** steht im Programmzähler, was zur Folge hat, daß im Programm zu dieser Zeile „gesprungen“ wird. Wenn es keine Programmzeile mit dieser Marke gibt, wird das Programm abgebrochen.

je M

Effekt:

Wenn die Nullflagge **zf** gesetzt ist, d. h. den Wert 1 hat, steht die Nummer der ersten Programmzeile mit der Marke **M** im Programmzähler, andernfalls die Nummer der folgenden Programmzeile. Als nächste wird im Programm die entsprechende Zeile ausgeführt.

`jne M`

Effekt:

Wenn die Nullflagge `zf` gelöscht ist, d. h. den Wert 0 hat, steht die Nummer der ersten Programmzeile mit der Marke `M` im Programmzähler, sonst die Nummer der folgenden Programmzeile.

`jc M`

Effekt:

Wenn die Ausleihflagge `cf` gesetzt ist, steht die Nummer der ersten Programmzeile mit der Marke `M` im Programmzähler, andernfalls die der folgenden Programmzeile.

`jnc M`

Effekt:

Wenn die Ausleihflagge `cf` gelöscht ist, steht die Nummer der ersten Programmzeile mit der Marke `M` im Programmzähler, andernfalls die der folgenden Programmzeile.

`cmp R`

Effekt:

Wenn der Wert des Registers `R` mit dem des Akkus `ax` übereinstimmt, ist die Nullflagge `zf` gesetzt, andernfalls gelöscht, und wenn der Wert von `R` kleiner als der von `ax` ist, ist die Ausleihflagge `cf` gesetzt, andernfalls gelöscht.

`clc M`

Effekt:

Die Ausleihflagge `cf` ist gelöscht, d. h. hat den Wert 0.

`stc M`

Effekt:

Die Ausleihflagge `cf` ist gesetzt, d. h. hat den Wert 1.

`cmc M`

Effekt:

Die Ausleihflagge `cf` ist komplementiert, d.h. ist gesetzt, falls sie vorher gelöscht war, und umgekehrt.

`push R`

Effekt:

Der Wert des Registers `R` ist auf dem Stapel abgelegt.

`pop R`

Effekt:

Wenn der Stapel keinen Wert enthielt, wird das Programm mit einer entsprechenden Fehlermeldung abgebrochen. Andernfalls enthält das Register `R` jetzt den obersten Wert des Stapels und dieser Wert ist vom Stapel entfernt.

`call M`

Effekt:

Die Nummer der ersten Programmzeile mit der Marke `M` steht im Programmzähler was zur Folge hat, daß im Programm zu dieser Zeile „gesprungen“ wird. Die nächste Rückkehranweisung hat zur Folge, daß der Programmzähler die Nummer der nächsten Programmzeile (die auf die Zeile mit der `cal`-Anweisung folgt) enthält.

`ret`

Effekt:

Das Programm bzw. das Unterprogramm ist beendet.

Für den Namen des verwendeten Registers – hier `R` – kann in den Anweisungen die Nummer eines beliebigen Registers eingesetzt werden; entsprechendes gilt für die verwendete Marke `M`. In diesem Sinne sind diese Zeilen als *Schablonen* für Anweisungen zu verstehen.

Beispiele

Das folgende Programm berechnet die Fakultät des Wertes des Registers `b` und schreibt sie in das Register `a`, wenn `a` anfangs den Wert 1 enthält:

```
    lda a
A   mul b
    sta a
    dec b
    jne A
    ret
```

Dieses Beispiel ist allerdings nur korrekt, wenn der Wert von `b` nicht 12 überschreitet, weil das Ergebnis sonst $\geq 10^9$ ist.

Etwas weiter arbeitet das folgende Miniprogramm:

```
A   lda a
    mul b
    sta a
    stb c
    lda d
    mul b
    add c
    sta d
    dec b
    jne A
    ret
```

Das Ergebnis mod 10^9 (d. h. die 9 niedrigen Stellen) steht nach der Ausführung des Programms im Register `a`, das Ergebnis div 10^9 (d. h. die bis zu 9 hohen Stellen) im Register `d`.

Die Leserinnen und Leser sollten sich durch Nachrechnen davon überzeugen, daß dieses Miniprogramm bis zum Startwert 19 von `b` korrekt ist.

DAS SIMULATIONSPROGRAMM MINI

Das Programm **Mini**

Mini ist ein Simulationsprogramm zur Ausführung von Miniprogrammen. Für die Bezeichner in Miniprogrammen, deren Ausführung von **Mini** simuliert werden soll, müssen folgende Konventionen eingehalten werden:

- *Variable*, d. h. Namen von Registern, werden mit einem kleinen Buchstaben (von **a** bis **t**) bezeichnet,
- *Marken* mit einem Großbuchstaben (von **A** bis **Z**).

Allgemeines zur Programmbedienung

Die Bedienung des Programms ist denkbar einfach.

Neben den Buchstaben-, Ziffern- und Zeichentasten zum Eingeben von Text werden einige Sondertasten zur Korrektur von Eingaben und zur Steuerung des Programmablaufs gebraucht.

Der Eingabekorrektur dienen die folgenden Tasten:

- die Rückschritt- \leftarrow und die Entfernungstaste **Entf** zum Löschen einzelner Zeichen, in Kombination mit der Umschalttaste \uparrow zum Löschen des Eingabefeldes,
- die Pfeiltasten \leftarrow und \rightarrow nach links und rechts sowie
- die Anfangstaste **Pos1** und die Endetaste **Ende** zum Bewegen im Text.
- Mit der Einfügetaste **Einfg** wird zwischen dem Einfüge- und dem Überschreibemodus umgeschaltet, wobei der aktuelle Modus an der unterschiedlichen Cursorform erkennbar ist: ein Unterstrich im Einfüge- und ein rechteckiger Block im Überschreibemodus.

Der Programmablauf wird mit

- der Eingabetaste \leftarrow , der Schlußtaste **Esc**, der Rückschritt-Taste \leftarrow ,
 - den Pfeil- \uparrow und \downarrow und den Bildtasten **Bild \uparrow** und **Bild \downarrow** nach oben und unten sowie
 - der Tabulatortaste \Leftrightarrow gesteuert;
- gelegentlich in Verbindung mit der Umschalttaste \uparrow .

Fehlermeldungen (in gelber Schrift auf rotem Grund) werden mit der Rücktaste \leftarrow quittiert.

Hinweise zur Arbeit mit Mini

Die Dateinamen von Miniprogrammen *müssen* immer die Endung `.mini` haben.

- Mit einem beliebigen Editor ein Miniprogramm editieren, z. B. `test.mini`
- **Mini** durch Eingabe von `Mini` aufrufen, wobei der Name des auszuführenden Miniprogramms (ohne die Endung `.mini`) als Parameter mitgegeben werden kann, z. B. `Mini test`,
- den Namen des Miniprogramms eingeben und mit der Eingabetaste `↵` bestätigen
- ggf. das Miniprogramm editieren (der eingebaute Editor ist aber noch nicht besonders leistungsfähig, deshalb kann zum Editieren längerer Miniprogramme auch ein Editor eigener Wahl benutzt werden)
- den Editiermodus mit der Schlußtaste `Esc` verlassen
- die Startwerte der verwendeten Register eingeben
- mit der Eingabetaste `↵` schrittweise durch das Miniprogramm laufen (die Registerinhalte werden dabei laufend angezeigt),
- falls gewünscht, mit `Esc` den Schrittmodus verlassen und das Miniprogramm bis zum Ende durchlaufen lassen,
- falls gewünscht, die Programmausführung von **Mini** mit der Kombination `Strg + C` abbrechen und
- nach Ausführung des Programms (was mit der Meldung `Programm ausgeführt` quittiert wird) **Mini** mit `Esc` beenden.

ANHANG

Aufgaben

Entwickeln Sie Miniprogramme zur Berechnung folgender Zahlen
testen Sie sie mit **Mini**:

- Potenz zweier Zahlen
- Summe von drei oder mehr Zahlen
- Minimum/Maximum von drei oder mehr Zahlen
- ggT und kgV zweier Zahlen
- Addition, Multiplikation zweier Brüche
- Produkt von drei oder mehr Zahlen
- Summe, Differenz zweier *großer* Zahlen ($\geq 10^{18}$)
- $26!$
- 2^{100}
- Produkt zweier großer Zahlen
- Quotient großer Zahlen
- Binomialkoeffizienten
- Fibonacci-Zahlen
- Anzahl der Suchschritte bei binärer Suche in einer Menge von n Elementen ($n < 10^9$)
- ... das gleiche, wenn n eine große Zahl ist
- ...