1

1. Das Programm

1.1 Programmzeilen

```
          _name     Mein~Programm
;         ...
          mov       r0        a
m1:
          add       r0        10                  ; plus 10
          mul       r0        pi
          div       r0        180
;         ...
          _end
```

RTA-Programme bestehen – wie alle Programme – aus einzelnen Zeilen. Jede Zeile ist ein Befehl. Die erste Zeile eines Programmes soll ein `_name`-Befehl sein, die letzte Zeile ein `_end`-Befehl. Dazwischen stehen beliebige Befehle.

Die einzelnen Zeilen bestehen aus Token. Token bestehen aus aufeinanderfolgenden Druckzeichen. Token werden durch ein oder mehrere Leerzeichen oder Tabulatoren (white spaces) voneinander getrennt. Token sind also z. B. `_name` `Mein~Programm` `mov` `r0` `a` `pi` `180`.

In Assembler trennt man Token üblicherweise durch Tabulatoren. So gliedert sich der Quelltext in einzelne Spalten. Jede Spalte ist 8 Zeichen breit.

1.2 Programmspalten

Die Markenspalte

Die erste Spalte ist die Markenspalte.

```
m1:
          add       r0        10                  ; plus 10
```

Hier wird eine Marke „m1“ durch die Markendefinition „m1:“ definiert. Weiteres zu Marken siehe unten.

Anmerkung: In RTA ist eine Markendefintion zugleich ein Markenbefehl, und muss einzeln auf einer separaten Zeile stehen. Ein anderer Befehl auf dieser Zeile ist verboten. In anderen Assemblern können Zeilen wie `m1: add r0 10` zulässig sein.

Die Befehlsspalte

Computer arbeiten Befehle ab. Assembler programmieren heißt, die Befehle in der Reihenfolge hintereinander aufschreiben, wie sie im abgearbeitet werden sollen. Jeder Befehl steht auf einer Zeile. Die Befehle haben bestimmte feste Befehlscodes, z. B. `add, sub, mul, div`:

```
m1:
        add     r0      10              ; plus 10
```

Hier ein add-Befehl, der eine Addition von 10 auf eine Variable r0 anweist.

Die Operandenspalten

Ein Befehl braucht Zahlen, mit denen er rechnet. Das sind die Operanden. RTA-Befehle können bis zu 3 Operanden haben, die in den 3 Spalten hinter der Befehlsspalte stehen:

```
m1:
        add     r0      10      .       ; plus 10
```

Hier wird z. B. zu einer Variablen mit dem Namen „r0“ die Zahl 10 hinzuaddiert. Der Punkt `.` steht hier für einen nicht benutzten Operanden und kann weggelassen werden.

Kommentar und ähnliches

Kommentar: Alles, was nach einem Semikolon `;` folgt, ist Kommentar. Kommentar wird von der Maschine nicht mitgelesen. Kommentare beginnen vorzugsweise nach 5 Tabulatoren in der Kommentarspalte:

```
m1:
        add     r0      10              ; plus 10
```

Kommentarzeilen: Eine Programmzeile kann auch ausschließlich Kommentar enthalten. Dann steht das Semikolon ganz vorn:

```
;
; Programm
;
        mov     a       b
```

Explizite Zeilenverkettung: Drohen Zeilen zu lang zu werden, so kann man sie in RTA mit dem Zeilenverkettungszeichen `¶` („Paragraph“) auf mehrere Zeilen verteilen. Immer dann, wenn ein `¶`

als letztes Zeichen einer Zeile unmittelbar vor dem Zeilenumbruch notiert wird, ist die nächste Zeile die Fortsetzung der vorangegangenen Zeile:

```
        input    a          Meridianstreifenabstand_in_Bog¶
                            enminuten
```

Das ¶-Zeichen, der unmittelbar folgende Zeilenumbruch, sowie in der folgenden Zeile führende Leerzeichen und Tabulatoren werden ignoriert. `Meridianstreifenabstand_in_Bogenminuten` ist folglich ein Token.

Anmerkung: Für das Zeichen ¶ gebe man bei gedrückter Alt-Taste „0182“ auf dem Ziffernblock ein, Kurzschreibweise hierfür auch <Alt/0182>.

Leerzeilen: Eine oder mehrere Programmzeile(n) können auch ganz leer sein:

```

```

1.3 Operanden

Die hinter den Befehlscodes stehenden Operanden geben an, *mit welchen Daten* gerechnet werden soll. Die Operanden heißen in RTA *Symbole.* Alle Symbole werden in einer Symboltabelle verzeichnet. In dieser Tabelle gibt es eine Spalte für Symbolnamen und eine Spalte für Symbolwerte:

Symboladresse	Symbolname	Symbolwert
1	a	300
2	R0	0
3	alpha	0
4	P23	-15

Der ***Symbolname*** ist das Token, welches im Programmtext steht. Der ***Symbolwert*** ist ein Speicherplatz, auf welchem eine Zahl gespeichert und mit dem gerechnet werden kann. Die Zeilennummer der Tabelle heißt ***Symboladresse.***

RTA kennt keine Datentypen. Es gibt somit keine Unterscheidung von Integerzahlen, Gleitkommazahlen, Zeichenketten etc. Um es vorwegzunehmen: RTA rechnet alles in *double float.*

Allerdings kann ein Symbol auf 4 verschiedene Arten *genutzt* werden: als Variable, als Zahl, als Marke oder als Zeichenkette.

Variablen

In den meisten Programmiersprachen müssen Variablennamen grundsätzlich aus Buchstaben bestehen, denen oft Ziffern und Unterstriche folgen können. In RTA ist jede Kombination von

Druckzeichen zugelassen. Das können Buchstaben wie **A, B, C, a, b, c,** Umlaute wie **ä, ö, à, ø** oder Sonderzeichen, wie **:, -, +, (, :** sein – ja es sind sogar Exotica wie **¨, ©, º, ¿, ¢** oder **±** zulässig. Ziffern wie **0, 1, 2, 3** sind natürlich ebenfalls erlaubt. So sind z. B. Token wie

```
a        aa       r0       1f1e1    ..
Lücke    MAß      â        ()       z1_in_m
M12      M(33)    $er      alpha    ,,,
ad       m        z2       ¤¥       a100
```

gültige Variablennamen. Man kann also auch Variablen

```
a'       b"       delta²   A(%)
```

nennen. RTA unterscheidet Groß- und Kleinbuchstaben. So bezeichnen die drei Symbole

```
alpha    Alpha    ALPHA
```

drei verschiedene Symbole. Sobald eine Variable erstmalig auftaucht, wird sie in die Symboltabelle eingetragen. Es gibt also keine Deklarationspflicht für Symbole.

Zahlen

Zahlen sind in RTA ebenfalls ganz normale Variablen.

```
12       -123     +3.14E12
```

Wenn der RT-Assembler ein neues Symbol erstmals auffindet, so versucht er den Symbolnamen als *Zahl* zu interpretieren. Das Ergebnis wird dann sofort als Symbolwert eingetragen. Auf diese Art erhält ein Symbol mit dem Namen `1` den Wert 1 und ein Symbol mit dem Namen `10` den Wert 10. So erhalten Zahlen-Symbolnamen „ihre eigenen“ Zahlenwerte zugewiesen.

Zahlen werden so, wie in Programmiersprachen üblich, notiert. Das Dezimaltrennzeichen ist der Punkt `.`, nicht das Komma. Token `3.141592653` erzeugt Symbolwert 3,141592653. Man sollte nicht `3,141592653` schreiben. Dann wird lediglich Symbolwert 3 erzeugt, da beim Komma abgebrochen wird. Die Zeichen `-` und `+` können Vorzeichen sein. Das Exponentialkennzeichen `E` oder `e` hilft, große und kleine Zahlen kürzer zu notieren. `1E6` steht für den Wert 1 Million und `25e-4` für den Wert 0,0025.

Marken

Mitunter müssen Positionen im Programm markiert werden, z. B. um Sprungsadressen festzulegen. Dies erfolgt mit einer Markendefinition.

Eine Markendefinition ist ein Token an einem Zeilenbeginn mit abschließendem Doppelpunkt. Die Markendefinition

```
m001:
```

trägt ein Symbol „m001“ in die Symboltabelle ein. Als Symbolwert wird die Zeilennummer im Programm eingetragen, in welcher sich die Markendefinition befindet.

Wie man sieht, gehört der Doppelpunkt selbst nicht zum Symbolnamen (siehe hierzu auch unten, Befehl `_lab`). Auf Markendefinitionszeilen dürfen in RTA keine (anderen) Befehle stehen.

Zeichenketten

Mitunter braucht man einen Text. Hierfür nutzt RTA den *Symbolnamen*. Der Befehl

```
        input    a        Anzahl
```

nutzt das Token `Anzahl` als Zeichenkette *Anzahl*.

Weil Leerzeichen Token trennen, können diese in Zeichenketten nicht auftreten. Hier gilt nun die Tilde `~` als Leerzeichen-Stellvertreter.

Weiterhin dient der Backslash `\` als Zeilenumbruch-Stellvertreter. In dem Befehl `pause` steht z. B. die Notation

```
        pause    Programm~fortsetzen?\(Fortsetzen-Taste~drücken)
```

für einen auszugebenden Mitteilungstext

Programm fortsetzen?
(Fortsetzen-Taste drücken)

In RTA gibt es also nicht die Zeichenketten-Form `"abc def ghi"`, oder `'abc def ghi'`, wie in anderen Programmiersprachen. Dies erscheint zunächst etwas gewöhnungsbedürftig, ist aber einfach und verständlich. Es gibt es auch keine Spezialnotationen, wie `\\`, `\"`, `\'`, `\n`, `\t`, `\0112`,

`\0xa3` etc., die oft wechseln und sich von Programmiersprache zu Programmiersprache fein unterscheiden und alles durcheinanderbringen.

Oft ist es sinnvoll, Zeilenverkettungszeichen und Zeilenumbruch-Stellvertreter zu kombinieren:

```
pause    Programm~fortsetzen?\¶
         („Fortsetzen"-Taste~drücken)
```

Das verbessert die Lesbarkeit, man beachte aber, dass die Zeichen `\` und `¶` völlig unterschiedliche Bedeutungen haben:

• Das `\`-Zeichen ist der Zeilenumbruchs-Stellvertreter. Es ist Tokenbestandteil und Zeichen *innerhalb* einer Zeichenkette. Es ersetzt einen Zeilenumbruch *im* Symbolnamen.

• Das `¶`-Zeichen ist der explizite Zeilenverketter. Es weist als letztes Druckzeichen einer Programmzeile an, den folgenden Zeilenumbruch zu ignorieren und mit den auf der nächsten Zeile folgenden Druckzeichen fortzusetzen. Es ist *kein* Tokenbestandteil.

1.4 Vordefinierte Variablen

Zum Programmstart stehen eine Reihe von Variablen automatisch bereit und können sofort genutzt werden. Dies sind:

	Wert	Bedeutung
.	0	Das Leersymbol – immer mit dem Wert Null
..	1	Der Befehlszeiger mit der Nummer des aktuellen Befehls
pi	3,14159265358979	Die Kreiszahl π
pi/2	1,5707963267949	π/2
pi/4	0,785398163397448	π/4
e	2,71828182845906	Die Basis der natürlichen Zahlen *e*
®	6371004,2029572	Die Erdumfang in Metern
®f	0	Die Abplattung des Erdellipsoids
°(	0,0174532925199433	Ein Faktor, der Gradmaß in Bogenmaß umrechnet
(°	57,2957795130824	Ein Faktor, der Bogenmaß in Gradmaß umrechnet
eps	1E–99	Die kleinste Zahl, die sich auf Null addieren lässt
max	9.999999999999E+99	Die größte zulässige Zahl
r0	0	]
r1	0	]
r2	0	]
r3	0	] 8 zur beliebigen freien Nutzung zur Verfügung
r4	0	] stehende „Register“
r5	0	]
r6	0	]
r7	0	]

Von besonderer Bedeutung ist das Leersymbol. Es hat den Namen „**.**“, ist nicht beschreibbar und wird, wenn es gelesen wird, immer mit dem Wert Null gelesen. Wenn ein Befehl zu wenig Operanden enthält, so wird für diese das Leersymbol eingesetzt. Dies führt meistens zu einem sinnvollen Standardverhalten.

Der Befehlszeiger hat den Namen „**..**“ und enthält immer die Codeadresse
(= Programmzeilennummer) des gerade abgearbeiteten Befehls.

Alle anderen vordefinierten Variablen sind wie gewöhnliche Variablen beliebig les- und schreibbar. Man darf also z. B. die Variable „e“ nicht verändern, wenn man zugleich das e als Basis der natürlichen Logarithmen nutzen will.

Hinweis: Die Vimage-Implementation von RTA kennt noch einige weitere vordefinierte Variablen, wie z. B. **`x x' y y'`** und den Erdradius. Dies wird hier nicht weiter behandelt.