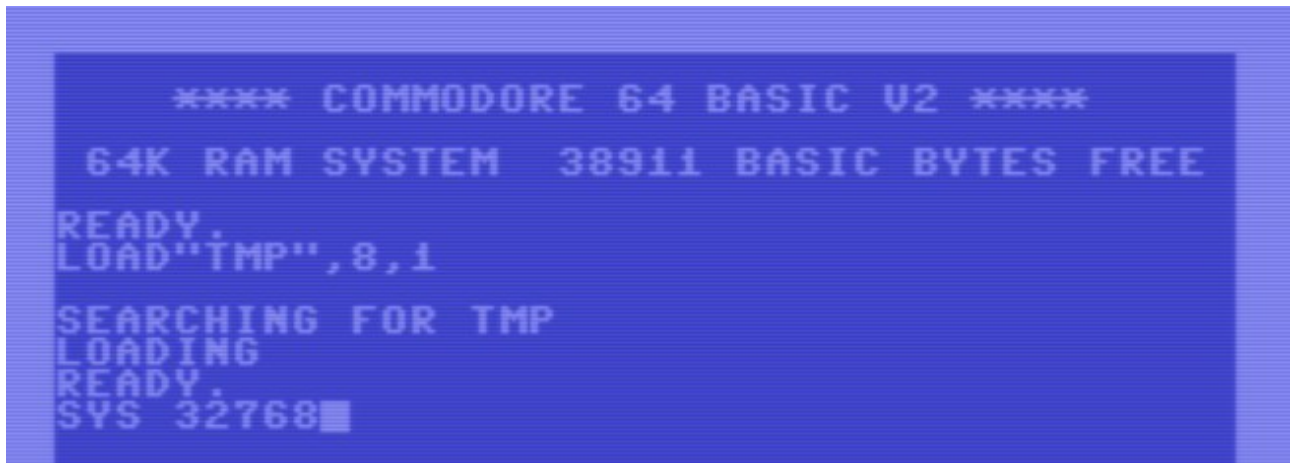


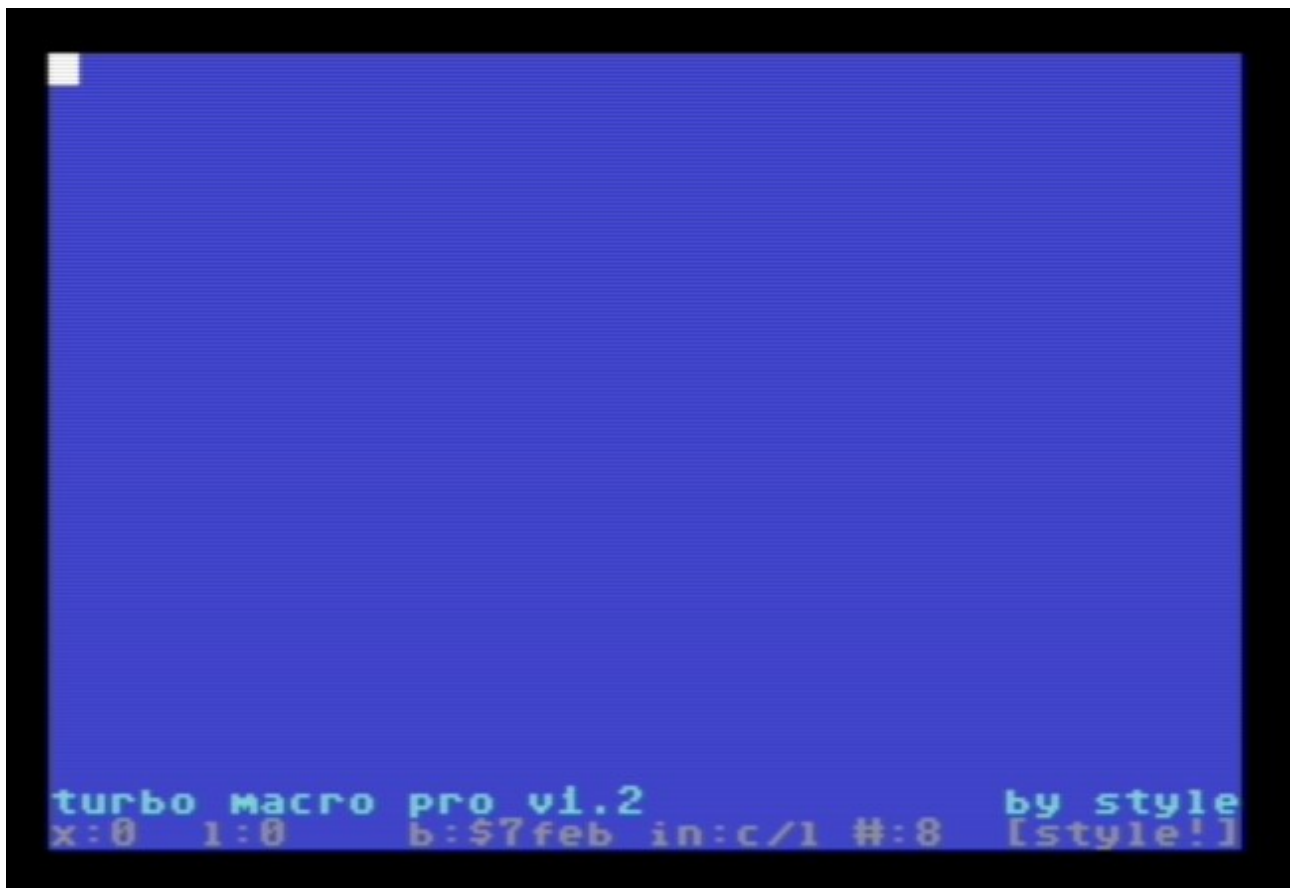
## Einführung in die Arbeit mit Turbo Macro Pro

Um mit dem Turbo Macro Pro arbeiten zu können, müssen wir ihn natürlich zuerst von der Diskette in den Arbeitsspeicher laden.

Legen Sie also die Diskette mit dem Label TMP ins Laufwerk und laden den TMP (so werde ich ihn ab jetzt abgekürzt benennen) mit der Anweisung LOAD „TMP“,8,1 in den Arbeitsspeicher.

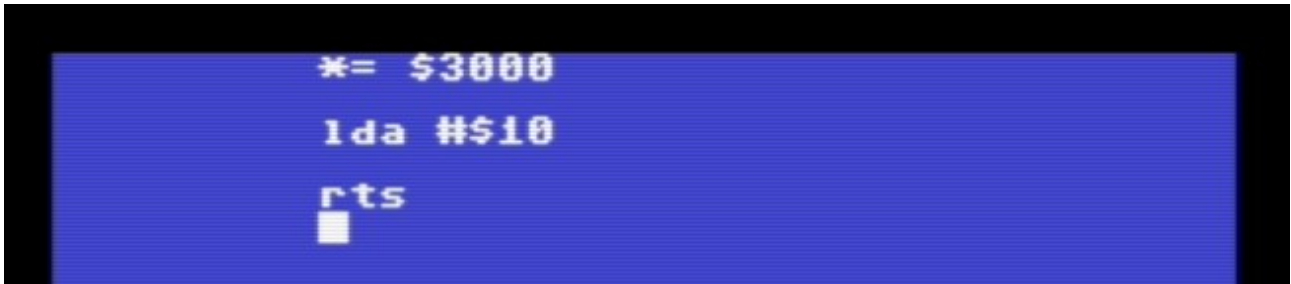


Nachdem Sie den TMP mit SYS 32768 gestartet haben, sollte sich dieser mit folgender Anzeige präsentieren.



## Eingabe, Assemblierung und Starten eines Assembler-Programms

Beginnen wir also mit einem ganz einfachem Programm und geben folgende Anweisungen ein:



Die Anweisung '\* = \$3000 legt fest, dass das Programm beginnend bei der Adresse \$3000 im Arbeitsspeicher abgelegt werden soll.

Es folgen zwei einfache Befehle, deren Bedeutung ich Ihnen sicher nicht erklären muss :)

Wie können wir dieses Programm nun starten? Da dieses zum jetzigen Zeitpunkt nur als Text im Editor vorliegt, muss es zunächst einmal assembliert und in Form von Maschinenbefehlen ab der Adresse \$3000 in den Speicher geschrieben werden.

Der TMP wird über Tastenkombinationen gesteuert, wobei die meisten durch Drücken der Taste, welche den nach links weisenden Pfeil darstellt, eingeleitet werden.

Auf einem echten C64 ist folgende Taste gemeint:

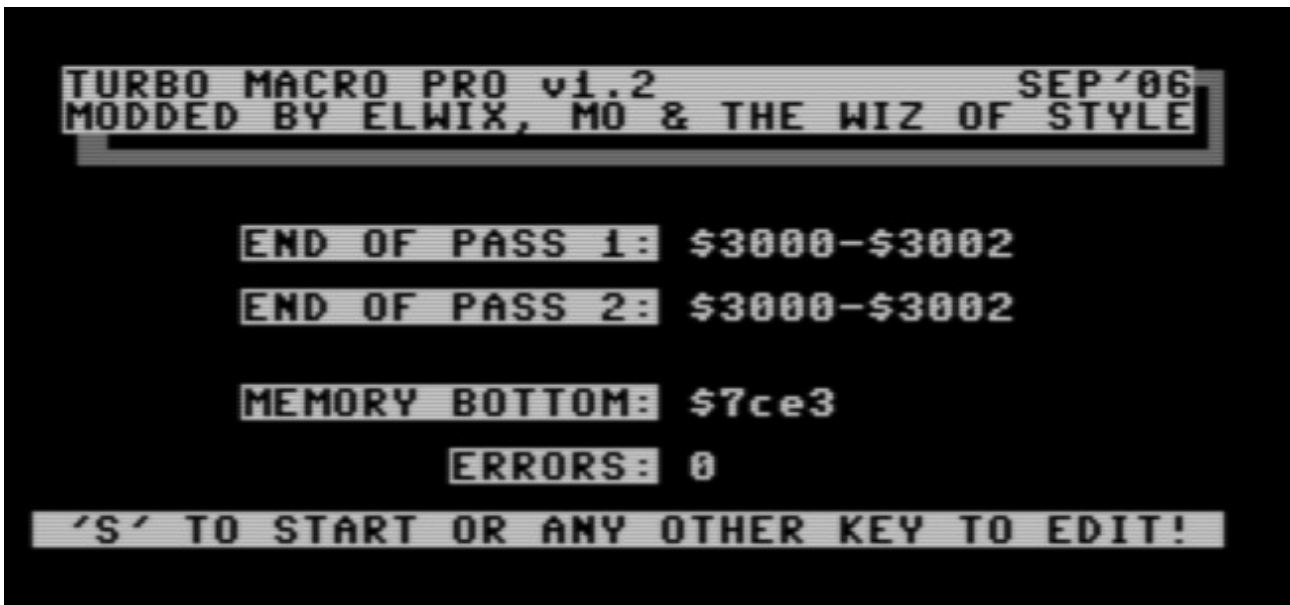


Falls Sie mit einem Emulator, wie z.B. dem VICE arbeiten, dann müssen Sie stattdessen die Taste drücken, auf der das Fragezeichen abgebildet ist:



Um das Programm zu assemblieren, müssen wir die Pfeil-Taste gefolgt von der Taste 3 drücken.

Daraufhin sollte folgende Meldung erscheinen:



Hier wird die Start- und Endadresse unseres Programms angezeigt, es belegt im Speicher also die Adressen von \$3000 bis \$3002.

An den Adressen \$3000 und \$3001 befindet sich der Maschinencode für den Befehl LDA #\$10 und an Adresse \$3002 der Maschinencode für den Befehl RTS.

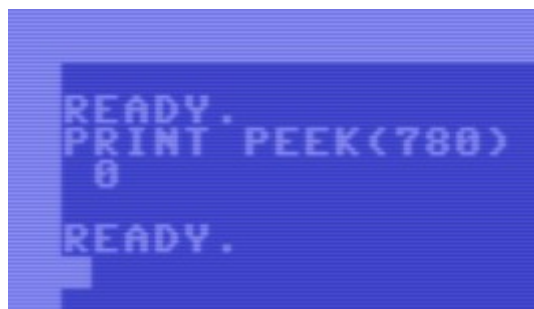
Fehler wurden, wie nicht anders zu erwarten, keine gefunden.

Nun befindet sich unser Programm also als ausführbarer Maschinencode im Speicher. Hier wird angezeigt, dass man das Programm durch Drücken der Taste S starten kann.

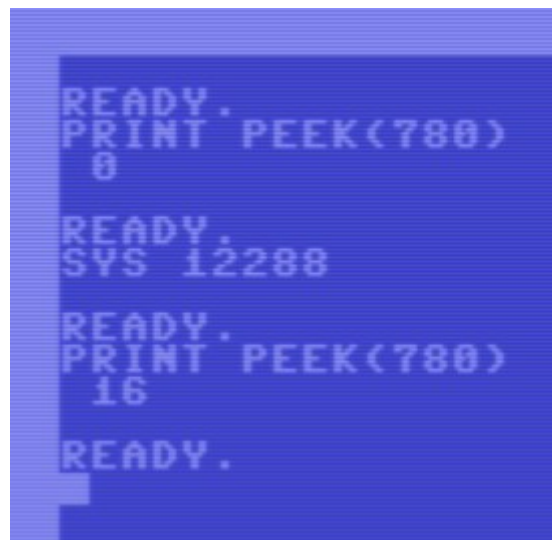
Aus welchen Gründen auch immer, scheint dies jedoch nicht wie erwartet zu funktionieren.

Es findet zwar ein Wechsel zu Basic statt, aber wenn das Programm gestartet worden wäre, müsste der Akkumulator den Wert 16 enthalten.

Ein Auslesen der Speicherstelle 780 zeigt jedoch ein anderes Bild:



Erst wenn das Programm manuell durch Eingabe des Befehls SYS 12288 gestartet wird, stimmt der Inhalt des Akkumulators.



```
READY.  
PRINT PEEK(780)  
0  
  
READY.  
SYS 12288  
  
READY.  
PRINT PEEK(780)  
16  
  
READY.
```

Seltsamerweise hat dies vereinzelt bei anderen Programmen (z.B. bei jenen aus dem später folgenden Kapitel über Sprites) funktioniert und dies sorgte bei mir für etwas Verwirrung.


Daher ziehe ich einen anderen Weg vor, um ein Programm auszuführen.

Ich drücke nicht die Taste S, um das Programm auszuführen, sondern irgendeine andere beliebige Taste, um wieder zum Editor des TMP zurückzukehren.

Dort drücke ich die vorhin angesprochene Pfeil-Taste und anschließend die Taste 1.

Dadurch findet ein Wechsel zu Basic statt und man kann das Programm ebenfalls durch Eingabe des entsprechenden SYS-Befehls starten. In diesem Fall hier durch den Befehl SYS 12288 und ich hatte bisher bei keinem einzigen Programm Probleme bei der Ausführung.

Durch Eingabe des Befehls SYS 32768 kann man jederzeit wieder zum TMP zurückkehren.



```
READY.  
PRINT PEEK(780)  
16  
  
READY.  
SYS 32768■
```

Es zeigt sich wieder der Editor mit unserem Assembler-Programm.

```
*= $3000
lda #$10
rts
█
```

Hier könnten wir nun Änderungen am Programm vornehmen und natürlich werden wir das auch tun :)

Fügen Sie nach dem Befehl LDA #\$10 daher die Befehle LDX #\$20 und LDY #\$30 ein.

```
*= $3000
lda #$10
ldx #$20
ldy #$30
█
rts
```

Drücken Sie nun die Pfeil-Taste gefolgt von der Taste 3, damit das geänderte Programm assembliert und in den Speicher geschrieben wird.

```
TURBO MACRO PRO v1.2          SEP '06
MODDED BY ELWIX, MO & THE WIZ OF STYLE

END OF PASS 1:  $3000-$3006
END OF PASS 2:  $3000-$3006

MEMORY BOTTOM:  $7cdd
ERRORS:  0

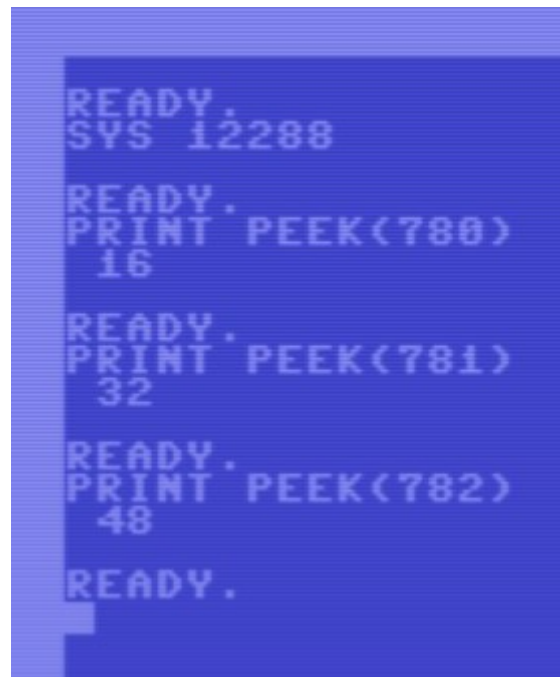
'S' TO START OR ANY OTHER KEY TO EDIT!
```

Hier sieht man, dass sich durch das Hinzufügen der beiden zusätzlichen Befehle die Endadresse des Programms verändert hat. Da sowohl der Befehl LDX #\$20 als auch der Befehl LDY #\$30 zwei Bytes im Speicher belegen, hat sich die Endadresse um vier auf \$3006 erhöht.

Wechseln Sie nun durch Drücken einer beliebigen Taste (außer der Taste S) zurück zum Editor.

Dort angekommen drücken Sie die Pfeil-Taste gefolgt von der Taste 1, um ins Basic zu gelangen.

Nun können Sie das Programm wiederum durch Eingabe des Befehls SYS 12288 starten.

A screenshot of a BASIC interpreter window with a blue background and white text. The text shows a sequence of commands and their outputs: 'READY.' followed by 'SYS 12288', then 'READY.' followed by 'PRINT PEEK(780)' and the output '16'. This is followed by 'READY.' followed by 'PRINT PEEK(781)' and the output '32'. Then 'READY.' followed by 'PRINT PEEK(782)' and the output '48'. Finally, it shows 'READY.' on a new line.

```
READY.  
SYS 12288  
  
READY.  
PRINT PEEK(780)  
16  
  
READY.  
PRINT PEEK(781)  
32  
  
READY.  
PRINT PEEK(782)  
48  
  
READY.
```

Dadurch wird das Programm ausgeführt und das Auslesen der Speicherstellen 780, 781 und 782 beweist, dass der Akkumulator, das X Register und das Y Register genau jene Werte enthalten, welche wir in unserem Programm angegeben haben.

Denn 16 entspricht dem hexadezimalen Wert \$10, 32 dem hexadezimalen Wert \$20 und 48 dem hexadezimalen Wert \$30.

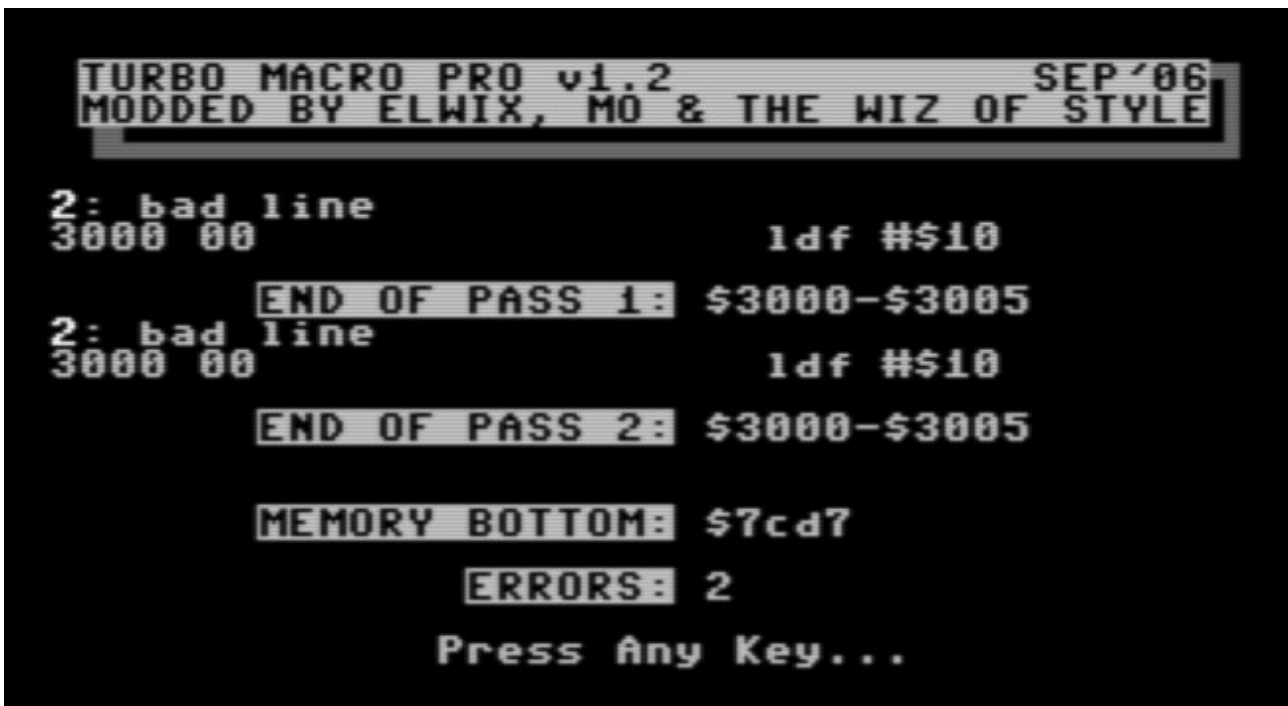
Sehen wir uns nun an, was passiert, wenn der TMP einen oder mehrere Fehler in unserem Programm gefunden hat.

Wechseln Sie also durch Eingabe von SYS 32768 wieder zum Editor des TMP und ändern den Befehl LDA in den unbekannten Befehl LDF:

A screenshot of a BASIC program listing window with a blue background and white text. At the top, it says '\*= \$3000'. Below that are four lines of code: 'ldf #\$10', 'ldx #\$20', 'ldy #\$30', and 'rts'. A small white square cursor is positioned to the left of the 'rts' line.

```
*= $3000  
  
ldf #$10  
ldx #$20  
ldy #$30  
■  
rts
```

Wenn Sie das Programm nun durch Drücken der Pfeil-Taste gefolgt von der Taste 3 assemblieren, wird der TMP folgende Fehlermeldungen anzeigen:



```
TURBO MACRO PRO v1.2 SEP '06
MODDED BY ELWIX, MO & THE WIZ OF STYLE

2: bad line
3000 00                                ldf #10

END OF PASS 1: $3000-$3005
2: bad line
3000 00                                ldf #10

END OF PASS 2: $3000-$3005

MEMORY BOTTOM: $7cd7
ERRORS: 2
Press Any Key...
```

Durch Drücken einer beliebigen Taste können wir nun zum Editor zurückkehren, um den Fehler zu korrigieren. Machen Sie also die vorhin durchgeführte Änderung wieder rückgängig und assemblieren das Programm durch Drücken der Pfeil-Taste gefolgt von der Taste 3 erneut.

Nun ist wieder alles in Ordnung und das Programm lässt sich auf dem vorhin beschriebenen Wege ausführen.

Fassen wir also die Schritte zusammen, die zur Erstellung und Ausführung eines Assembler-Programms im TMP nötig sind.

- Assembler-Programm eingeben
- Das Programm durch Drücken der Pfeil-Taste gefolgt von der Taste 3 assemblieren.

Falls der TMP Fehler im Programm meldet, eine beliebige Taste drücken, um in den Editor zurückzukehren und die Fehler zu korrigieren. Danach muss durch Drücken der Pfeil-Taste gefolgt von der Taste 3 eine erneute Assemblierung gestartet werden. Der Vorgang muss so oft wiederholt werden, bis vom TMP keine Fehler mehr gemeldet werden.

- Wenn der TMP keine Fehler meldet, eine beliebigen Taste außer der Taste S drücken, um wieder in den Editor zurückzukehren
- Durch Drücken der Pfeil-Taste gefolgt von der Taste 1 ins Basic wechseln
- Das Programm durch Eingabe des Befehls SYS gefolgt von der im Assembler-Programm festgelegten Adresse starten (im Falle von \$3000 also 12288, da die Adresse in dezimaler Form angegeben werden muss)

## Speichern eines Assembler-Programms auf Diskette

Bisher existiert unser Assembler-Programm nur als Text im Editor des TMP. Um das Programm jedoch dauerhaft zur Verfügung zu haben, müssen wir es auf einer Diskette speichern, damit wir es dann später wieder laden, starten oder ggf. auch ändern können.

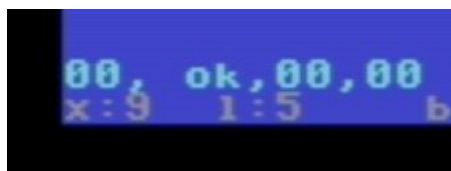
Um unser Assembler-Programm auf Diskette speichern zu können, müssen wir die Pfeil-Taste gefolgt von der Taste S drücken.

Dadurch erscheint am unteren Rand eine Abfrage nach einem Dateinamen, unter dem das Assembler-Programm gespeichert werden soll.



Geben Sie hier einen beliebigen Dateinamen, z.B. wie in diesem Fall hier den Namen tmptest1 ein und drücken die Return-Taste.

Wenn alles gut gegangen ist, wird am unteren Rand eine entsprechende Meldung angezeigt:





## Laden eines Assembler-Programms von Diskette

Gehen wir nun zurück zum Anfang, starten unseren C64 oder Emulator neu und laden den TMP.

Nachdem wir diesen durch Eingabe von SYS 32768 gestartet haben, sehen wir wieder den leeren Editor.

Um das Programm, welches wir vorhin unter dem Namen tmptest1 gespeichert haben, wieder zu laden, drücken Sie die Pfeil-Taste gefolgt von der Taste L.

Dadurch wird am unteren Rand eine Frage nach dem Namen der Datei angezeigt, die geladen werden soll. Geben Sie dort den Namen tmptest1 ein.



Nachdem Sie die Return-Taste gedrückt haben, wird das Programm wieder in den Editor geladen.



Am unteren Rand wird nun eine ok-Meldung angezeigt. Falls beim Laden ein Fehler aufgetreten ist, würde hier eine entsprechende Fehlermeldung angezeigt werden.

Angenommen, Sie wollen eine Datei laden, die nicht existiert, dann würde dies durch folgende Meldung angezeigt werden:



Nachdem wir unser Programm nun wieder geladen haben, können wir es auch wieder assemblieren und starten.

## Überschreiben eines bereits bestehenden Assembler-Programms auf Diskette

Da wir unser Programm nach dem Laden beliebig verändern können und diese Änderungen natürlich auch in die Datei auf der Diskette übernehmen wollen, gilt es beim erneuten Speichern der Datei eine Besonderheit zu beachten.

Wenn wir Änderungen an unserem Programm vorgenommen haben, müssen wir beim anschließenden Speichern der Datei die Zeichenfolge '@:' vor dem Dateinamen einfügen.

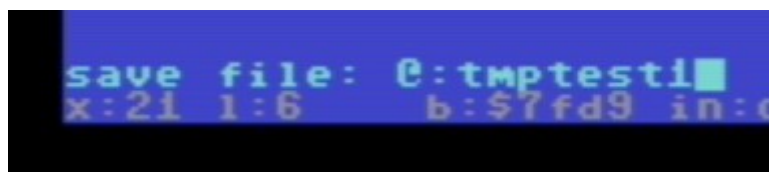
Dadurch wird die bereits bestehende Datei auf der Diskette überschrieben und enthält nun die von uns eventuell vorgenommenen Änderungen.

Fügen Sie also beispielsweise den Befehl STA \$0400 ein:



```
*= $3000
lda #$10
ldx #$20
ldy #$30
sta $0400
rts
```

Wenn Sie nun diese Änderung in die bereits bestehende Datei tmptest1 übernehmen wollen, dann müssen Sie anschließend bei der Abfrage des Dateinamens die Eingabe wie folgt ändern:




```
save file: @:tmptest1
x:21 1:6      b:$7fd9 in:c
```

Bewegen Sie nun den Cursor hinter die letzte 0 der Adresse \$0400, welche dem Befehl STA folgt, und entfernen durch mehrmaliges Drücken der Backspace Taste den Befehl STA \$0400 wieder aus dem Programm.

Laden Sie nun die Datei tmptest1 auf die vorhin beschriebene Weise durch Drücken der Pfeil-Taste gefolgt von der Taste L und anschließender Eingabe des Dateinamens tmptest1.

Sie werden sehen, dass der STA Befehl in die Datei übernommen wurde und nun wieder im Editor angezeigt wird.

Wenn Sie die Zeichenfolge @: vor dem Dateinamen nicht angeben, dann erhalten Sie die Meldung, dass die Datei bereits existiert:



```
63, file exists,00,00
x:9 1:6      b:$7fd9 in:
```

Die Änderungen werden in diesem Fall nicht in die bestehende Datei übernommen.

### **Wichtige Anmerkung:**

Egal ob Sie ein völlig neues Programm eingeben oder Änderungen an einem bereits bestehenden Programm vornehmen – die oberste Regel lautet immer: Vor dem Starten des Programms IMMER den aktuellen Stand des Programms auf Diskette speichern!

Gerade bei der Assembler-Programmierung kommt es sehr oft vor, dass es zu Abstürzen des C64 kommt und man dadurch keine Gelegenheit mehr hat, in den Editor des TMP zurückzukehren und den aktuellen Stand des Programms auf Diskette zu sichern.

In diesem Fall war dann die gesamte Arbeit umsonst und Sie müssen mit der Eingabe von vorne beginnen. Dies kann, je nach Umfang des Programms und der durchgeführten Änderungen, natürlich sehr viel Zeit verschlingen und ziemlich frustrierend sein.

Nun beherrschen Sie bereits die grundlegenden Schritte, die nötig sind, um sinnvoll mit dem TMP arbeiten zu können.

Es gibt jedoch noch eine ganze Menge weiterer Tastenfunktionen, die Ihnen bei der Arbeit gute Dienste leisten können.

### **Erstellen eines eigenständigen Programms aus einem Assembler-Programm**

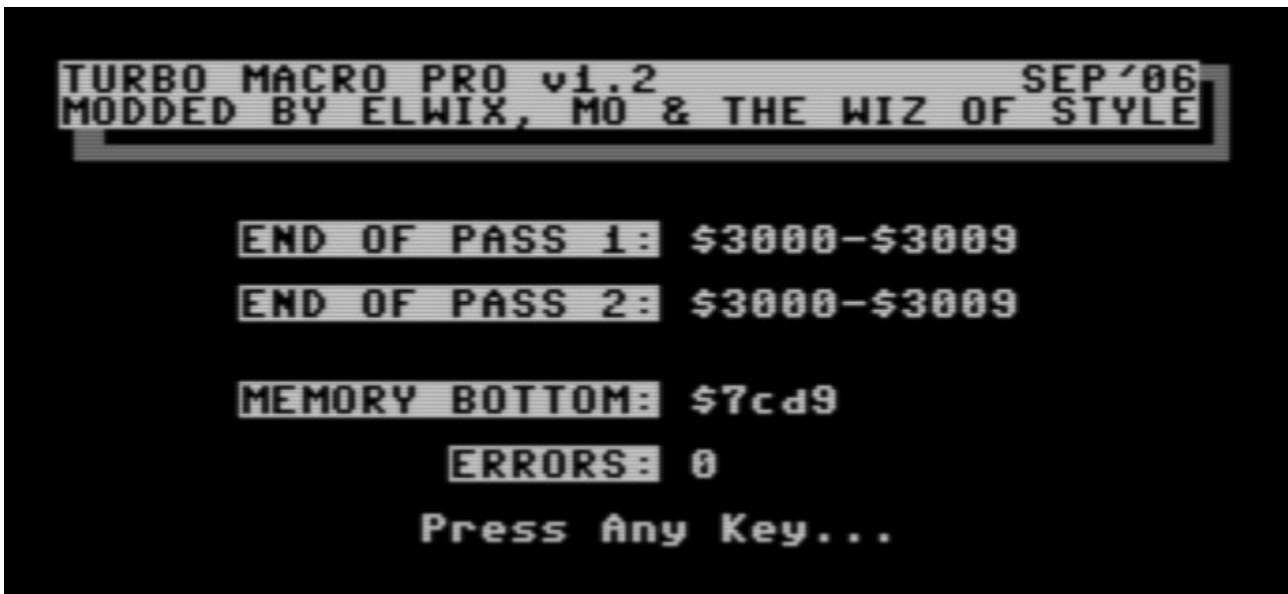
Nachdem man ein Assembler-Programm erstellt und erfolgreich getestet hat, will man im Normalfall daraus natürlich eine Datei erstellen, die man ohne Zuhilfenahme des TMP von der Diskette laden und ausführen kann.

Der TMP bietet diese Funktion durch Drücken der Pfeil-Taste gefolgt von der Taste 5 an. Daraufhin wird am unteren Rand die Frage nach einem Dateinamen angezeigt, unter dem das eigenständige Programm auf der Diskette gespeichert werden soll.



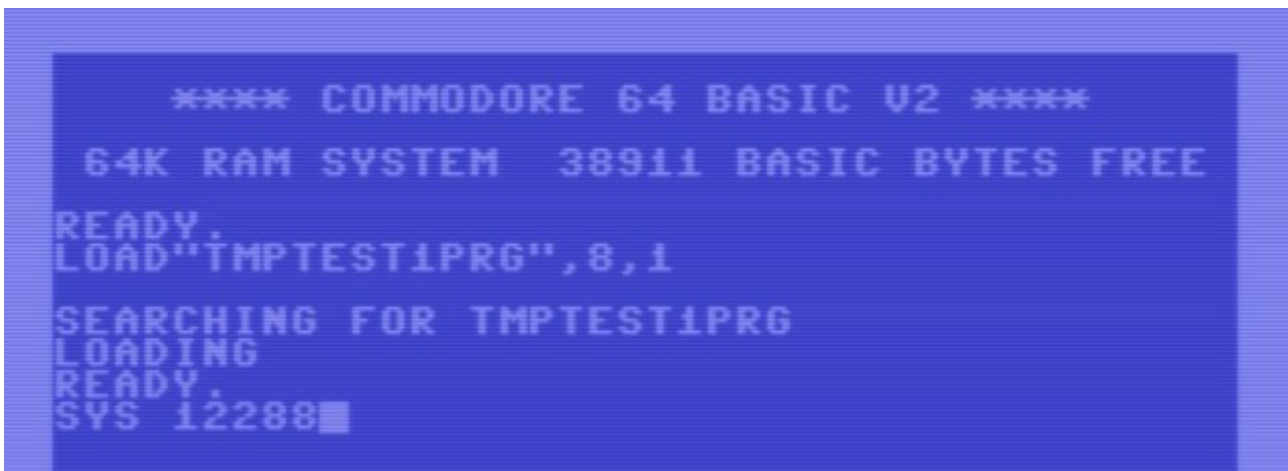
Geben Sie in unserem Fall den Namen tmptest1prg ein und Drücken die Return-Taste.

Daraufhin wird folgende Meldung angezeigt:



Wenn alles, wie in diesem Fall (0 Errors), gut gegangen ist, wird die Datei tmptest1prg auf der Diskette erstellt.

Starten Sie nun den C64 neu, laden das Programm wie folgt von der Diskette und starten es mit dem Befehl SYS 12288.



Nach dem Start des Programms wird in der linken, oberen Ecke des Bildschirms der Buchstabe P ausgegeben. Dies wurde durch den neu hinzugefügten Befehl STA \$0400 bewirkt und durch Eingabe von PRINT gefolgt von PEEK's auf die Speicherstellen 780, 781 und 782 zeigt sich, dass der Akkumulator, das X Register und das Y Register die gewünschten Werte beinhalten.

```

P
  **** COMMODORE 64 BASIC V2 ****
  64K RAM SYSTEM  38911 BASIC BYTES FREE
READY.
LOAD"TMPTEST1PRG",8,1
SEARCHING FOR TMPTEST1PRG
LOADING
READY.
SYS 12288

READY.
PRINT PEEK(780),PEEK(781),PEEK(782)
      16          32          48
READY.

```

Angenommen, wir würden nun den SMON starten und ab Adresse \$3000 disassemblieren, dann sehen wir, dass unser Programm, wie erwartet, ab der Adresse \$3000 im Speicher zu finden ist.

```

.D 3000
,3000  A9  10      LDA  #10
,3002  A2  20      LDX  #20
,3004  A0  30      LDY  #30
,3006  8D  00  04  STA  0400
,3009  60          RTS
-----
.■

```

Wir können das Programm im SMON natürlich auch durch Eingabe des Befehls G 3000 starten.

Zuvor müssen wir jedoch den Befehl RTS in den Befehl BRK ändern, damit der SMON nach Beendigung des Programms nicht verlassen wird. Würden wir es beim Befehl RTS belassen, würde ja ein Sprung ins Basic stattfinden.

Diese Änderung können wir natürlich, wie bereits von unserer Arbeit mit SMON bekannt, direkt im SMON durchführen, indem wir den Cursor auf den Befehl RTS bewegen und stattdessen den Befehl BRK eingeben.

Anzeige nach dem Start des Programms:

```

PG 3000
PC  SR  AC  XR  YR  SP  NU-BDIZC
;300A 30 10 20 30 F6 00110000
.■

```

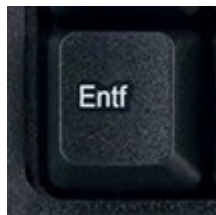
In der linken, oberen Ecke sieht man wieder den Buchstaben P und die Inhalte des Akkumulators (AC), des X Registers (XR) und des Y Registers (YR) entsprechen auch den erwarteten Werten \$10, \$20 und \$30.

## **Löschen von einzelnen Zeilen**

Wollen Sie eine einzelne Zeile in Ihrem Assembler-Programm löschen, dann bewegen Sie den Cursor in diese Zeile und drücken die Pfeil-Taste gefolgt von der INST/DEL Taste, wenn Sie auf einem echten C64 arbeiten.



Falls Sie im Emulator arbeiten, dann drücken Sie die Taste, auf der das Fragezeichen zu sehen ist, gefolgt von der Taste mit der Aufschrift „Entf“.



Die Zeile, in der sich der Cursor befindet, wird gelöscht und die darunterliegenden Zeilen rücken um eine Zeile nach oben.

## **Kommentare**

Durch Kommentare wird die Lesbarkeit eines Programms stark verbessert. Auch der TMP bietet die Möglichkeit, Programme mit Kommentarzeilen zu versehen. Dazu muss man der jeweiligen Zeile ein Semikolon voranstellen.



Angewandt auf unser kleines Programm vom Beginn dieser Einführung, könnte dies dann so aussehen:



```

; programm beginnt bei
; adresse $3000
; start von basic aus mit
; sys 12288

*= $3000

; akkumulator mit dem wert
; $10 (dezimal 16) laden

lda #$10

; programm beenden

rts

```

turbo macro pro v1.2 by style  
x:9 l:15 b:\$7f56 in:c/l #:8 [style!]

Bei solch einfachen Befehlen sind im Normalfall keine Kommentare nötig, aber es sollte hier ja auch nur gezeigt werden, wie man Kommentare in ein Assembler-Programm integrieren kann.

## Einfügen von Trennzeilen

Manchmal nutzt man Kommentarzeilen, um Programmteile optisch voneinander abzugrenzen.

Damit man diese Kommentarzeilen nicht selber, Zeichen für Zeichen, eingeben muss, bietet der TMP ebenfalls eine entsprechende Tastenkombination über die Pfeil-Taste gefolgt von der Taste 2 an.

Ich habe nachfolgend zur Veranschaulichung eine solche Trennzeile nach dem Befehl RTS eingefügt.



```

; programm beenden

rts
-----

```



## Blockoperationen

Manchmal ist es notwendig, nicht nur eine einzige Zeile, sondern einen ganzen Block, welcher aus mehreren Zeilen besteht, zu löschen, zu kopieren oder zu verschieben.

Unabhängig davon, ob man einen Block löschen, kopieren oder verschieben will, muss man klarerweise zunächst den Beginn und das Ende des Blocks festlegen.

Ich habe im Editor folgende fünf Kommentarzeilen eingegeben:

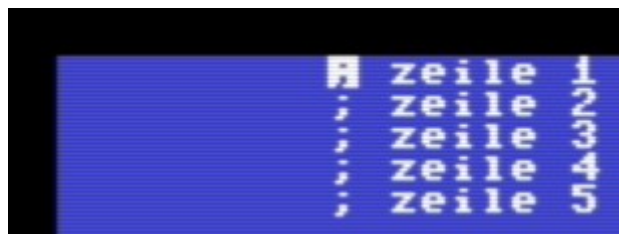


```
; zeile 1
; zeile 2
; zeile 3
; zeile 4
; zeile 5
```

x:9 l:5 b:\$7fbc in:c/l #:8 [style!]

Angenommen, wir möchten nun aus den ersten drei Zeilen einen Block bilden.

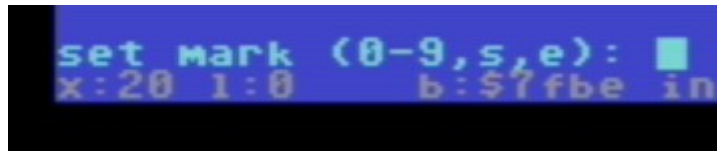
Als erstes bewegen wir den Cursor auf den Strichpunkt in Zeile 1.



```
; zeile 1
; zeile 2
; zeile 3
```

Dann drücken wir die Pfeil-Taste gefolgt von der Taste M.

Am unteren Rand erscheint nun folgende Abfrage:

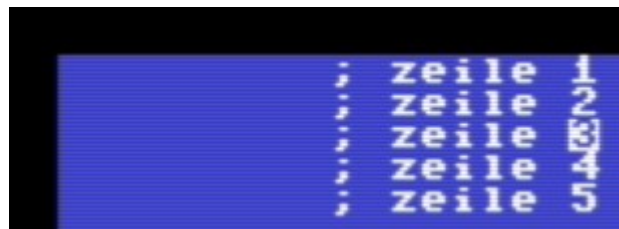


```
set mark (0-9,s,e): ■  
x:20 l:0 b:$7fbc in
```

Der TMP bietet hier durch die Angabe 0-9 offensichtlich die Möglichkeit, mit mehreren Blöcken zu arbeiten. Ich habe diese Möglichkeit jedoch noch nie genutzt und werde daher auch nicht näher darauf eingehen und mich stattdessen auf einen einzigen Block beschränken.

Hier drücken wir die Taste S, wodurch der Beginn des Blocks festgelegt wird.

Nun bewegen wir den Cursor auf die 3 in Zeile 3.



```
; zeile 1  
; zeile 2  
; zeile 3  
; zeile 4  
; zeile 5
```

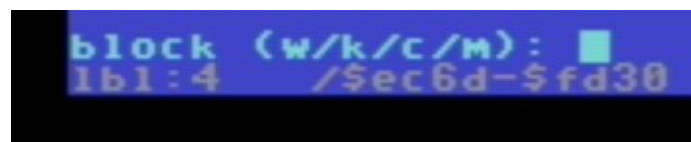
Nun drücken wir wieder die Pfeil-Taste gefolgt von der Taste M. Die obige Abfrage erscheint erneut, nur dieses mal drücken wir die Taste E, wodurch nun auch das Ende des Blocks festgelegt wird.

Nun wollen wir diesen Block aus drei Zeilen direkt unterhalb von Zeile 5 anfügen. Dazu bewegen wir den Cursor direkt unter das Semikolon in Zeile 5.



```
; zeile 1  
; zeile 2  
; zeile 3  
; zeile 4  
; zeile 5
```

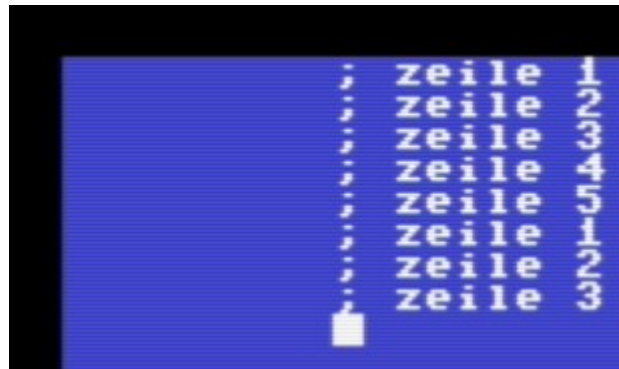
Nun drücken wir die Pfeil-Taste gefolgt von der Taste B, woraufhin am unteren Rand folgende Abfrage erscheint:



```
block (w/k/c/m): ■  
lbl:4 /$ec6d-$fd30
```

Hier haben wir die Möglichkeit, auszuwählen, ob wir den Block in eine Datei schreiben, ihn löschen, kopieren oder verschieben wollen. Da wir den Block kopieren wollen, drücken wir hier die Taste C, woraufhin die drei Zeilen kopiert werden.

Inhalt des Editors nach dem Kopiervorgang:



Als nächstes wollen wir denselben Block, also wiederum die ersten drei Zeilen, verschieben. Dazu legen wir, wie vorhin beschrieben, zunächst den Beginn und das Ende des Blocks fest und bewegen den Cursor wieder direkt unter das Semikolon in Zeile 5.

Nun drücken wir wieder die Pfeil-Taste gefolgt von der Taste B, woraufhin wieder die vorherige Auswahl-Abfrage erscheint.

Dieses mal drücken wir die Taste M, da wir den Block verschieben wollen.

Inhalt des Editors nach dem Verschieben des Blocks:



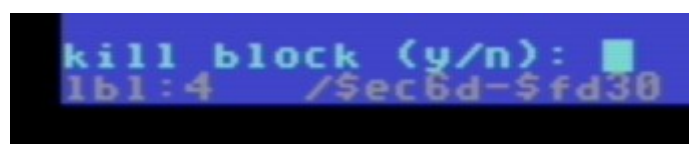
Hier sieht man, dass die obersten drei Zeilen nun am oberen Rand verschwunden sind und unter den zuvor kopierten Block verschoben wurden.

Nun will ich Ihnen noch zeigen, wie man einen Block löschen kann. Ich möchte dies anhand der obersten zwei Zeilen (zeile 4 und zeile 5) demonstrieren.

Dazu definieren wir diese beiden Zeilen, wie zuvor beschrieben, als Block und drücken dann die Pfeil-Taste gefolgt von der Taste B.

Es erscheint wieder die Abfrage, welche Aktion man ausführen möchte und dieses mal drücken wir die Taste K, da wir den Block löschen wollen.

Vor dem tatsächlichen Löschen erscheint noch eine Sicherheitsabfrage:



Diese bestätigen wir durch Drücken der Taste Y und nach dem Löschen des Blocks, sollte sich im Editor folgendes Bild zeigen:



Wie Sie sehen können, sind die beiden Zeilen mit den Inhalten zeile 4 und zeile 5 verschwunden.

### Schnell-Navigation innerhalb des Assembler-Programms

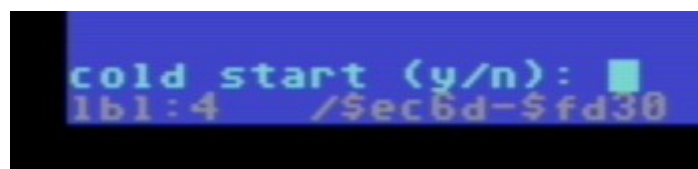
Funktionstaste	Wirkung
F1	Bewegt den Cursor um 20 Zeilen nach oben
F2	Bewegt den Cursor zum Beginn des Programms
F3	Bewegt den Cursor um 200 Zeilen nach oben
F5	Bewegt den Cursor um 200 Zeilen nach unten
F7	Bewegt den Cursor um 20 Zeilen nach unten
F8	Bewegt den Cursor zum Ende des Programms

### Einen Kaltstart des TMP durchführen

Durch Drücken der Pfeil-Taste gefolgt von der Taste C können Sie einen Kaltstart des TMP durchführen, d.h. er wird auf jenen Zustand zurückgesetzt, in dem er sich direkt nach dem Laden von der Diskette befand.

Das aktuell im Editor befindliche Assembler-Programm wird dadurch natürlich gelöscht.

Wegen der drastischen Auswirkungen wird nach dem Drücken der Tastenkombination noch eine Sicherheitsabfrage angezeigt, ob man den Kaltstart wirklich durchführen will.



Nun beherrschen Sie die wichtigsten Funktionen, um mit dem TMP erfolgreich Assembler-Programme erstellen bzw. ausführen zu können und sie werden sehen, dass das Programmieren im Vergleich zum SMON, vor allem bei größeren Programmen, viel mehr Spaß macht.