E65C02EM

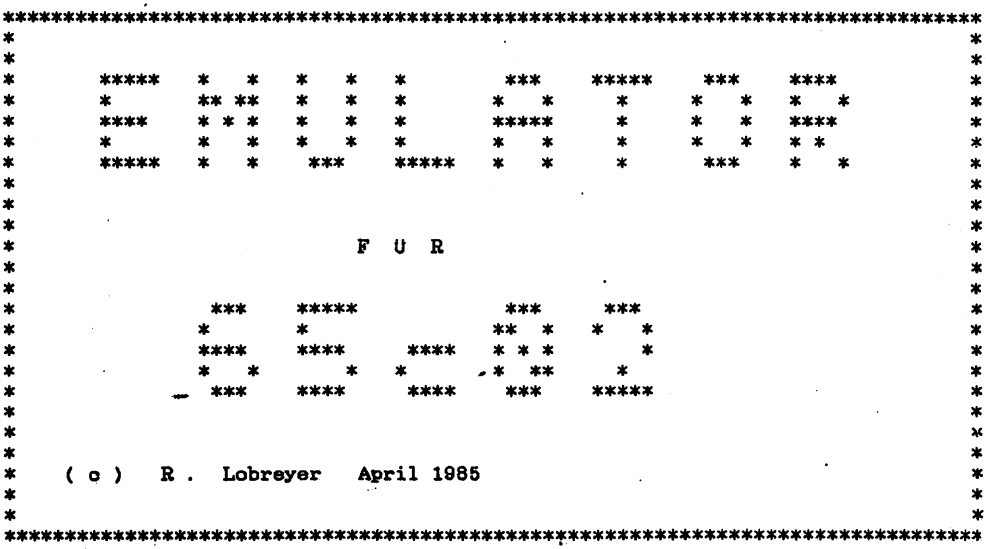
Ein Emulationsprogramm für den Mikroprozessor 65C02 auf dem NDR-Computer

Graf Elektronik Systeme GmbH
Magnusstraße 13 8960 Kempten/Allg.Tel.(0831)6211 Teletex 831804

Filiale Kempten
Computervilla
Ludwigstraße 18b
8960 Kempten/Allg.

Filiale Hamburg
Ehrenbergstraße 56
2000 Hamburg 50
Tel. (040) 388151

Filiale München
Georgenstaße 61
8000 München 40
Tel. (089) 2715858



I N H A L T S Ü B E R S I C H T

0. Vorbemerkung

Als ich anfing dieses Programmpaket zu erstellen, war mir zuerst der Gedanke eines Apple-Emulators in den Sinn gekommen, was aber durch die vielen Ein/Ausgabe-Adressen des Apple II in Bezug auf Geschwindigkeit unerträglich geworden wäre. Ich habe mich deshalb entschlossen nur den Prozessor 6502 bzw. den neueren 65c02 zu emulieren (emulieren heißt eine vorhandene Hardware durch Software zu simulieren). Nach der Fertigstellung des Emulator-Moduls musste noch die Möglichkeit geschaffen werden, Programme leicht erstellen und austesten zu können. Hierzu wurde ein Disassembler und ein Assembler erstellt. Nachdem die Textgröße auf über 140K-Byte angestiegen war, aber immer noch ca. 4K-Byte auf einem der beiden EPROM's brach lagen, wurde der Entschluß gefasst etwas umfangreichere Testmöglichkeiten zu schaffen, als Sie sie vielleicht schon vom 68008 Einzelschritt her gewohnt sind. Auch wurde eine Schnittstelle zum Grundprogramm eingerichtet, um auch die spezifischen Ein/Aus-gabe-Fähigkeiten des NDR-Klein-Computers ausnutzen zu können.

Auf dem Softwaremarkt wird derzeit ein Z80/CPM-80 Emulator angeboten, welcher unter CP/M 68K einen Z80 Prozessor emuliert. Die Emulationsgeschwindigkeit entspricht der eines mit ca. 1.3 MHZ getaktetem Z80 Prozessor. Eine solch hohe eschwindigkeit ist nur dann möglich wenn, die I/O Tätigkeiten etc. ohne Zwischenschritt durch den 68000 Prozessor ausgeführt werden. Diese Taktfrequenz wird bei meinem Programm nicht erreicht. Nach einigen Tests habe ich eine virtuelle Taktfrequenz von ca. 0.1 MHZ ermittelt. Dies ist auch dem Umstand zuzuschreiben, daß beim 6502 die Flags des Statusregister sehr viel öfter beeinflußt werden, als beim Z80.

Ich hoffe dennoch, daß Sie mit dem Erwerb dieses Software-Paketes zufrieden sind. Sollten Sie Verbesserungsvorschläge etc. haben, so wenden Sie sich bitte an mich, denn aus Fehlern kann man nur lernen!

Mit freundlichen Grüßen

Rolf Lobreyer

P.S.: Dieses Handbuch kann die einzelnen Befehle der 6502/65c02 CPU nicht erklären. Es wird deshalb dringend die Anschaffung weiterer Bücher, die sich mit dem 6502/65c02 beschäftigen, empfohlen !

80 Konfiguration und Inbetriebnahme

I. Konfiguration

Zum Betrieb kann jeder NDR-Klein-Computer mit CPU68K als Zentraleinheit, Grundprogramm ab Version 3.1 und mindestens 16K-RAM oberhalb des Grunprogramms eingesetzt werden. Die Ein- und Ausgaben erfolgen über die Standard-Ein/Ausgaben :
Tastatur mit Baugruppe KEY und Bildschirm mit Baugruppe GDP64K.

Die Minimal-Konfiguration für einen störungsfreien Betrieb des Programmes auf einen Blick :

(i) CPU68K
(ii) ROA64 mit Grundprogramm ab Version 3.1 und mindestens 16 KByte RAM
(iii) KEY + Tastatur
(iv) GDP64K + Bildschirm

II. Einsetzen der beiden EPROMS auf einer ROA64-Karte

Zum Betrieb dieses Programmpaketes sind die beiden beigefügten 8 KByte Eproms in zwei aufeinanderfolgende Sockel einer ROA64 Karte einzustecken, wobei darauf zu achten ist, daß die Einkerbung eines jeden Eproms nach oben zeigt. (Eine Verpolung führt zur Zerstörung der Eproms !) Die Adresslage der ROA64-Karte sowie die Position der EPROMs auf der Karte ist für die Funktionstüchtigkeit unerheblich, da das Programm vollständig verschiebbar geschrieben ist.

Vorsicht!!! < Einsetzen der Eproms nur bei ausgeschaltetem Rechner>

Reihenfolge der Eproms beachten :

Eprom mit Aufschrift "E65C02EMU 0" zuerst einsetzen, dann das Eprom mit der Aufschrift "E65C02EMU 1" direkt in die rechts daneben-liegende IC-Fassung der ROA64-Karte stecken.

III. Test auf Funktionstüchtigkeit

Haben Sie die oben genannten Schritte vollzogen, steht einem Test nichts mehr im Wege. Stecken Sie die ROA-Karte auf den BUS zurück. Schalten Sie nun den Rechner wieder ein und drücken "W", um in das zweite Grundmenü zu gelangen. Rufen Sie die Option <Bibliothek> auf. Es sollte sich dabei folgendes Bild ergeben :

```
                    B I B L I O T H E K
 ---------------------------------------------------------
 !                                                       !
 !    NAME       :  EMULATOR                             !
 !    START      :  ______                               !
 !    LAENGE     :  003FFF                               !
 !                                                       !
 ---------------------------------------------------------
  Starten  J = JA
  cr = weiter , M = Menue
```

Anstatt _____ steht irgend eine Adresse. Sie spiegelt die Adresslage der Eproms im Adressbereich der Zentraleinheit wider. Der Start des "Emulator" kann nun durch drücken der Taste "J" erfolgen.
Es sollte ein Eingabefeld erscheinen, welches Sie auffordert eine RAM-Startadresse des 6502/65c02 anzugeben. Ist dies erfolgt (Adresse größer als $9000 wählen), so sollte sich das 65c02 Hauptmenu mit den weiter unten genannten Menu-Punkten melden. Verlassen Sie nun mit "W" das Menu, um den vorläufigen Test abzuschließen.

IV. Einzelkomponenten des Systems

Die Bedienerführung geschieht durch die Darstellung eines Menues, unter dessen Optionen ausgewählt werden kann. Als Menupunkte erscheinen :

1. Starten
2. Disassemblieren
3. Assemblieren
4. Einzelschritt
5. Ram-Start festlegen
6. Weiter

Die Bedeutung dieser Optionen wird hier nun kurz erklärt.

1. Starten :

Eingabeaufforderung einer Startadresse, ab der ein 6502/65c02 Programm steht das ausgeführt werden soll. (Programm liegt als HEXCODES vor !)

2. Disassemblieren :

Eingabeaufforderung einer Anfangs- und End-Adresse, zwischen der ein 6502/65c02 Programm disassembliert werden soll.

3. Assemblieren :

Ein sich im Speicher befindlicher Text wird durch den 65c02 Assembler des Emulators in 65c02 Maschinencode übersetzt.

4. Einzelschritt :

Wie unter 1. wird ein 6502/65c02 Porgramm ausgeführt, die Abarbeitung erfolgt aber in Einzelschritten, die auf dem Bildschirm protokolliert werden. Setzen bestimmter Optionen sind möglich.

5. Ram-Start festlegen :

Diese Option wird sowohl bei Eintritt in den Emulator aus der Bibliothek heraus, als auch unter Menupunkt 5 aufgerufen. Durch Eingabe einer Adresse, ab der sich RAM befinden sollte (max. 64K), wird die Adresse $0000 des 65c02 definiert. Auf diese Adresse beziehen sich alle weiteren Adresseingaben innerhalb des "Emulator-Betriebs".

In den weiteren Paragraphen erfolgt nun eine genauere Beschreibung der einzelnen Komponenten.

§1 Programmerstellung mit dem 65c02 Emulator

I. Eingabe von Maschinenbefehlen in Form von Sedezimal-Zahlen.
(sog. Maschinencode)
- Aufruf EMULATOR mit Hilfe der Bibliotheksfunktion
- Definieren der RAM-Start Adresse.
- Verlassen des 65c02 Menu's

Durch Eingabe der RAM-Start-Adresse, wird ein Symbol 'BASIS' in die Symboltabelle eingetragen, welches die aktuelle Adresse $0000 des 65c02 darstellt. So ist durch die Option AENDERN des 68008 Grundprogramms eine einfache Eingabemöglichkeit des 65c02 Maschinen-Code geschaffen.

Vorgehensweise:

Eingabe von 'BASIS + ' gewünschte Adresse, wobei Adresse eine Zahl im Bereich von 0 bis 65535 sein muss ($0 - $FFFF), da sich Adressenangaben beim 65c02 immer in diesem Bereich bewegen müssen.

Beispiel: Eingabe eines Maschinenprogramms ab Adresse $1000 :
- Wähle Option AENDERN
- Eingabe von 'BASIS + $1000' für Adresse
- jetzt gewünschtes Programm im Hexcode fortlaufend eingeben.
- mit M zum Menu zurückkehren.
- Jetzt Bibliothek anwählen und EMULATOR aufrufen.
- Das Programm kann nun unter 65c02 Menu-Punkt 2 durch Eingabe von $1000 gestartet werden.

Ein Nachteil dieses Verfahrens liegt darin, daß der Programmcode nur in Sedezimaler Form vorhanden ist, wodurch eine Änderung nur schwer möglich ist.
Ein weiterer Nachteil liegt wohl auch darin, dass das Programm erst einmal von Hand in Maschinencode übersetzt werden muss. (sehr fehleranfällig !) Deshalb ist die Übersetzung eines 6502 Porgramms durch einen Assembler oftmals vorzuziehen. Der Assembler bietet gegenüber der Methode ein Programm von Hand zu übersetzen einige Vorteile :
- Symbolische Befehls-Mnemonics anstatt der HEXCODES
- Als Sprungziele können Marken eingesetzt werden
- Die Sprungweiten werden durch den Assembler errechnet
- Fehler im Programm können schneller verbessert werden, da nun das Programm in Form eines Textes vorliegt.
- Das Programm wird überschaubarer, weil zusätzliche Kommentare an gefügt werden können.

II. Eingabe der Maschinenbefehle in Form von Assemblermnemonics

Wenn Sie sich bereits mit der Arbeitsweise des 68000/08 Assemblers vertraut gemacht haben, so werden Sie hier sicher keine Schwierigkeiten haben, da die Bedienung stark der des 68008 Assembler ähnelt.

Der 65c02 Assembler setzt einen sich im Speicher befindlichen Text, dessen Textstart dem Grundprogramm mitgeteilt werden muss, voraus. Soll der gerade aktuelle Text in Maschinencode übersetzt (assembliert) werden, dann ist folgendes zu beachten:

- Adressangaben sind nur im Bereich von $0 bis $FFFF erlaubt.
- Adressangaben beziehen sich immer auf die BASIS-Adresse des 65c02.
- EQU-Pseudoanweisungen sollten nur am Anfang eines Programmes vorkommen, da sonst Unstimmigkeiten mit der Adressrechnung auftreten können.
- Die während des Assemblerlaufs definierten Symbole, werden in diesselbe Symboltabelle eingetragen, wie Sie sie vom 68000 Assembler her kennen. Sollten Sie 68000 und 6502 Programme zur selben Zeit im Speicher haben, so ist eine eventl. Kollision der Bezeichner möglichst zu vermeiden. (6502 Symbole sind an der Zahl 2 oder 1, die dem Symbolnamen folgt, leicht zu erkennen.)
- Die Assembler-Optionen des Grundprogramms sind auch hier gültig.

Einzelheiten über die Assemblersyntax entnehmen Sie bitte dem Abschnitt über den Assembler.

Start des Assemblers:

Ist die Eingabe des Programmes im Texteditor beendet, so kann durch den Assemblerlauf der Test auf syntaktische Richtigkeit des Programmes erfolgen. Vor Start des Assembler kann eine Assembler-Option gewählt werden, so daß die Ausgabe z.B. auf einen Drucker etc. umgeleitet werden kann. Jetzt in der Bibliothek das Emulator-Menu anwählen. Wurde das Programm bis zu diesem Zeitpunkt noch nicht aufgerufen, so werden Sie zuerst zur Eingabe der RAM-Start-Adresse aufgefordert. (Sie muß größer als $9000 sein, da darunter zuwenig Platz für Variablen etc. vorhanden ist.) Wird die Eingabe der RAM-Start-Adresse gefordert, nachdem Sie bereits das 65c02 Menu aufgerufen hatten, so wurde diese Adresse durch den 68008 Assembler überschrieben und muß deshalb neu gesetzt werden.

($8160 ist die Adresse in der RAM-Start abgelegt wird. Sie liegt im oberen Teil des Eingabe-Buffer, wo sie durch den Assembler manchmal überschrieben wird. Bei verschobenem Grundprogramm entsprechend $8160 oberhalb des Grundprogrammes.)

Starten des Assemblers durch Anwählen des Menü-Punktes 'Assemblieren'.

§2 Der Assembler

Der Assembler gestattet es 65c02 Menmonics wie sie z.B. durch die Hersteller Rockwell, Commodore etc. vereinbart wurden in Maschinenbefehle umzusetzen.

Die hier vorgestellte Version ist ein 2 Pass Assembler, d.h. es erfolgen zwei Übersetzungsschritte ein und desselben Textes. Beim 1. Pass werden alle Befehle übersetzt und die auftretenden Symbole in eine Symboltabelle eingetragen. Weil einige Symbole während der Übersetzung noch nicht bekannt sind (sog. Vorwärtsreferenzen) müssen sie in den Operandenteil derjenigen Befehle, die solche Vorwärtsrefenzen als Operanden verwenden, (Zieladresse, Sprungadresse...) noch nachgetragen werden. Dieses Nachtragen, die Ausgabe des Textes und der Fehler geschieht im 2. Pass. (Streng genommen wird im 2. Pass der gesamte Text nochmals übersetzt und die nun definierten Symbole einfach als Operanden eingetragen.)

Der vorliegende Assembler wurde in 68000 - Maschinensprache geschrieben, ist demnach also ein Cross-Assembler, der Maschinenbefehle eines anderen Prozessors, der nicht selbst im Rechner steckt, übersetzt.

Es gelten diesselben Assembleroptionen wie beim 68000 Assembler des Grundprogrammes, welche durch einen gesonderten Menüpunkt im Grundprogramm angesprochen werden :

(i) "nur Fehlerausgabe auf Bildschirm ausgeben" gestattet es den Übersetzungsvorgang zu beschleunigen, da die Bildschirmausgabe abgeschaltet ist. (wird nur bei Fehlern wieder eingeschaltet)

(ii) "Ausgabe auf CRT" bedeutet, daß die vom Assembler erzeugte Ausgaben auf dem Bildschirm ausgegeben werden.

(iii) "Ausgabe auf LST" lenkt die Assemblerausgabe auf den Drucker um, der an die Centronics-Schnittstelle angeschlossen ist. Das gesamte Assemblerlisting wird auf einen Drucker ausgegeben.

(iv) "Ausgabe auf LST ohne LF" lenkt ebenfalls die Ausgabe auf den Drucker um, wobei jetzt aber nur das Zeichen <cr> = Wagenrücklauf gesendet wird. (Für Drucker, die bei <cr> auch ein Zeilenvorschub ausführen gedacht.)

2.1 Pseudobefehle

2.1.1 ORIGIN

Der ORG-Befehl des 6502 Assembler entspricht dem des 68000 mit dem kleinen Unterschied, daß ORG nur im Bereich von $0 bis $FFFF liegen kann. Bereichsüberschreitungen werden mit der entsprechenden Fehlermeldung quittiert. Die auf den ORG-Befehl folgende Adressangabe bezieht sich, wie alle weiteren Adressangaben im Assembler, auf die Anfangsadresse des 65c02. Wird 'ORG' weggelassen, so wird eine obligatorische Adresse von $1000 angenommen. Programme solten nicht vor Adresse $200 abgelegt werden, da sich sonst Unstimmigkeiten mit der Adressrechnung bei Sprüngen ergeben können. Im allgemeinen wird die Seite 0 des 65c02 als Speicher für Zeiger und Daten mit schneller Zugriffszeit benutzt, da die Operandenlänge nur ein Byte ist, statt der üblichen 2 Bytes bei absoluter Adressierung. Die Seite Null ist deshalb durch spezielle Adressierungsarten vor anderen ausgezeichnet und sollte nur als Daten-Speicher dienen. Ebenso sollte ein Programm nicht ab $100 abgelegt werden, da sich zwischen $100 und $1FF der System-Stack befindet, der auf diesen Bereich von 256 Bytes begrenzt ist.

```
Beispiel:  ORG  $400          ; Lege 6502-Code ab Adresse $400 ab
           LDA  #55           ; Lade Acku mit dezimal 55
           LDX   $35          ; lade X-Register mit  Inhalt
;                               Speicherzelle $35

           RTS
```

2.1.2 EQUAL

Durch den Pseudobefehl 'EQU' können Adressen oder Datenwerte mit bestimmten Symbolen-Namen benannt werden, unter dem sie dann in der Symboltabelle des Grundprogramms abgelegt werden. Als Werte der Equal-Definitionen sind sowohl 16-Bit-Adressen als auch 8-Bit Größen erlaubt. Wird dieser Wertebereich überschritten, so erfolgt auch hier die entsprechende Fehlermeldung. Als Wertdefinitionen sind auch arithmetische Ausdrücke erlaubt, die sich auf die vier Grundrechenarten der Ganzzahlarithmetik beschränken. Also z.B.
'AA EQU $2340+%10101010'
ist eine gültige Definition. Unter AA wird der Wert $23DA abgelegt. Bei der Definition ist noch folgendes zu Beachten :
EQU-Definitionen sollten nur zu Beginn eines Programmes erfolgen, da dann sichergestellt ist, daß der Assembler die richtige Adressierungsart für Befehle mit solchen Definitionen herausfinden kann.

2.1.3 DS.B/DS.W

DS.B bzw. DS.W bezeichnen Befehle zur Speicherplatz-Reservierung, wie dies für Operanden die im Speicher stehen notwendig ist. Äquivalente Befehle befinden sich auch beim 68000 Assembler, so daß eigentlich nichts weiter dazu bemerkt werden muss. Leglich die maximale Größe des zu reservierenden Bereichs ist auf 64K-Byte begrenzt !!

```
Beispiel:
DS.B   64          ; reserviere einen Speicherbereich von 64 Bytes
DS.W   64          ; reserviere einen Speicherbereich von 128 Bytes
```

2.1.4 DC.B/DC.W

Ein Assembler wäre wohl unvollständig, wenn er es nicht gestattete auch Konstantendefinitionen zuzulassen. DC.B steht für Deklariere eine Konstante mit Byte-Größe. Diese Konstante kann sowohl eine Zahl als auch ein Zeichen sein. Textkonstanten werden durch DC.B 'Text' deklariert. Zahl-Kostanten durch DC.B $.. ,DC.B .. oder durch DC.B %..... (mit . als korrekte Ziffer des Sedizimal-, Dezimal- oder Dual-System.) definert.
Sollen mehre Konstanten hintereinander im Speicher abgelegt werden, so können sie nacheinander, durch Komma getrennt, aufgeführt werden.

DC.W steht für Deklariere eine Konstante mit Wort-Größe. Als Wort werden hier nur Adressangaben akzeptiert, die sich im Bereich von 0 bis $FFFF befinden. Textkonstanten sind nicht erlaubt, da diese auch durch DC.B darstellbar sind. Die Einführung von DC.W hatte den Sinn, die Adressrechnung des 6502 zu entsprechen, da dieser Prozessor die Operanden der einzelnen Befehle nicht, wie beim 68000 und anderen Prozessoren üblich, zuerst das höherwertige und dann das niederwertige Byte aus dem Speicher liest, sondern gerade umgekehrt. DC.W berücksichtigt diesen Umstand, sodaß 16-Bit Konstanten entsprechend in umgekehrter Reihenfolge, also LSB zuerst dann MSB, in den Speicher abgelegt werden.

```
Beispiel:
DC.B 'Hallo dies ist ein Text',0     ; lege den Text als ASCII-Code
                                     ; in aufsteigender Reichenfolge
                                     ; in den Speicher ab. Zum Schluß
                                     ; noch $00 als Endemarkierung

DC.W $C000                           ; lege $00, dann $C0 in zwei
                                     ; aufeinanderfolgende Speicher-
                                     ; zellen ab
```

2.1.5 Least signifikant Byte / Most signifikant Byte

Wird die unmittelbare Adressierungsart (#..) verwendet, so ist es i.a. nicht möglich Adressen als Operanden einzusetzen, da diese 16 Bit groß sind. Durch die Einführung zweier Sonderzeichen ("<" und ">"), die nur bei der unmittlebaren Adressierungsart ihre Gültigkeit besitzen, wird dieses Problem umgangen. Man kann nun durch "#<"Adresse das LSB , also das Low-Byte, der Adresse oder durch "#>"Adresse das MSB, also das High-Byte, der Adresse in ein Register laden. Dies funktioniert auch für den Fall, daß die Adresse vor dem Assembliervorgang noch unbekannt ist. (Wenn Marken verwendet werden)

Beispiel:

```
LDX   #<TEXT          ; Lade niederwertiges Byte der Adresse  TEXT ins
                      ; X-Register
STX   $2003           ; lege niederwertiges Byte in Adresse $2003 ab
LDX   #>TEXT          ; Lade höherwertiges Byte der Adresse TEXT ins
                      ; X-Register
STX   $2004           ; lege höherwertiges Byte in Zelle $2004 ab.
RTS
TEXT:     DC.B  'Dies ist ein Beispiel',0
```

2.1.6 END

Durch 'END' kann dem Assembler mitgeteilt werden, wo sich das ENDE des zu assemblierenden Textes befindet. Nachfolgende Zeilen werden vom Assembler ignoriert. Ist 'END' im Programmtext nicht vorhanden, so wird der gesamte, sich im Speicher befindliche, Text assembliert.

2.2 MARKEN/LABELS

2.2.1 Definition

Sollen Marken/Labels im Programmtext definiert werden, so kann das durch Angabe von 'NAME EQU WERT' oder durch 'NAME:' erfolgen. Die EQU-Definition eines Labels wurde bereits unter 2.1.2 besprochen. Die zweite Möglichkeit wird oft auch als Define by Adress bezeichnet. Sie wurde vom 68000 Assembler entsprechend übernommen. Durch den nachfolgenden ':' wird gewährleistet, daß der Assembler eindeutig zwischen Befehl und Marke unterscheiden kann. So ist es auch möglich Befehlsnamen als Marken/Labels zu definieren ohne dadurch in Konflikt mit der 'Semantik' zu geraten. Für Labels mit nachfolgendem Doppelpunkt wird der gerade aktuelle Stand des Befehlszählers in die Symboltabelle eingetragen.

Beispiel:

```
ORG     $400            ; Programmcode ab Adresse $400 ablegen
NAME    EQU     $1000   ; Der Bezeichner NAME wird mit dem Wert
                        ; von $1000 in die Symboltabelle ein-
                        ; getragen.

START:  LDA     #$AA    ; Der Bezeichner START wird mit dem Wert
                        ; $400 in die Symboltabelle eingetragen
                        ; (der Befehlszähler ist noch auf $400)

FERTIG: RTS             ; Der Bezeichner FERTIG wird mit dem
                        ; Wert $402 in die Symboltabelle einget.
                        ; (Befehlszähler steht auf $402)
```

2.2.2 Unterschiede zum 68000 Assembler bei der Symbol-Definition

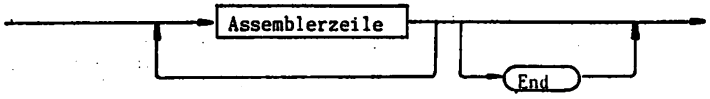
Im Gegensatz zu 68000 Symbolen sind alle Symbole, die durch den 65c02 Assembler in die Symboltabelle eingetragen werden, nur entweder Byte- oder Wort-Groß. Sie sind dadurch leicht von den Symbolen des 68000 zu unterscheiden, wenn im Grundprogramm der Menu-Punkt 'SYMBOLE' angewählt wird. 6502 Symbole sind dabei alle mit einer 0001 oder einer 0002 gekennzeichnet.

Wichtig !! Haben Sie mehrere Programme gleichzeitg im Speicher, so sollten alle Bezeichner unterschiedliche Namen tragen. Die Werte werden bei jedem Assemblerlauf (68000 oder 65c02) aktualisiert, sodaß bei gleichem Bezeichner derjenige Wert gerade gültig ist, der als letztes in einem Assemblerlauf definiert wurde, wodurch es daraus zu Ungereimtheiten kommen kann.

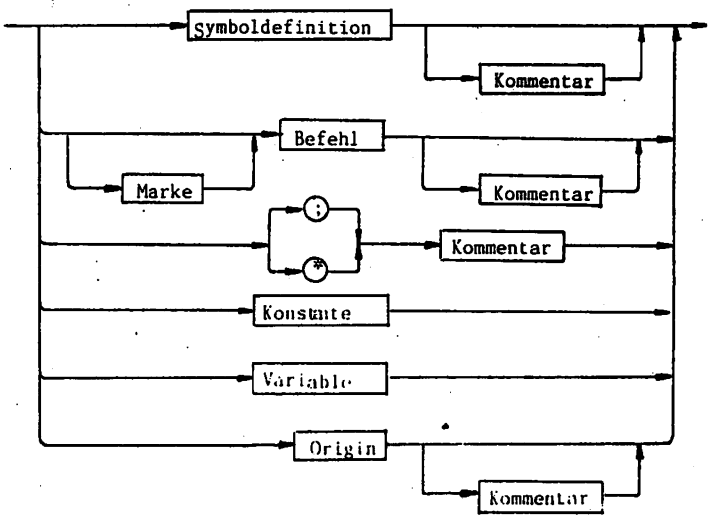
2.3 Assembler-Syntax

Die Assembler-Syntax soll durch Syntaxdiagramme, wie sie Sie vielleicht vom PASCAL/S her kennen, beschrieben werden. Die Eingabe des Programmtextes in Klein- und Groß-Buchstaben, auch gemischt, ist möglich.

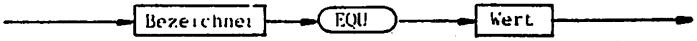
Programm



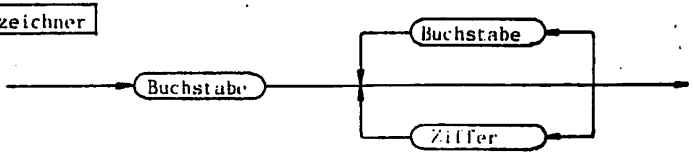
Assemblerzeile



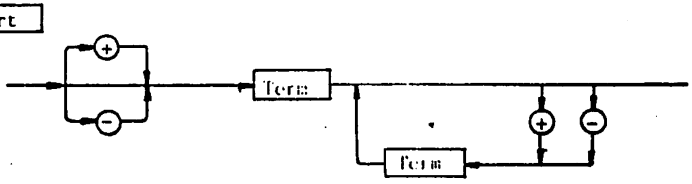
Symboldefinition



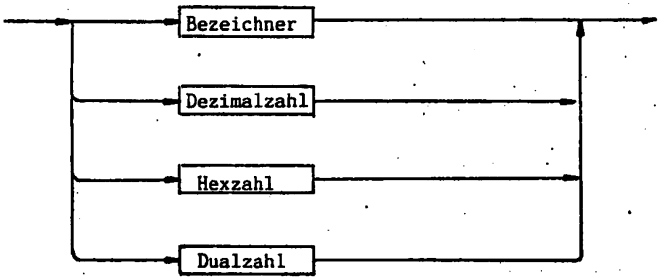
Bezeichner



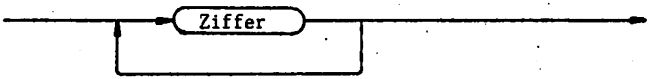
Wert



Term

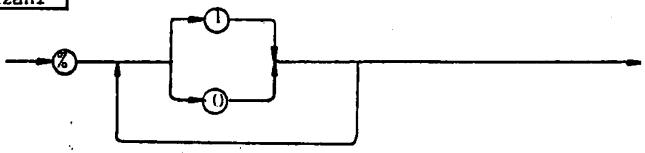


Dezimalzahl

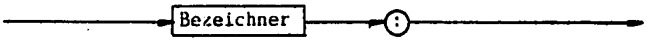


Hexzahl

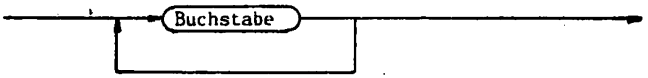
Dualzahl



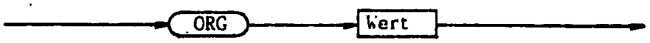
Marke



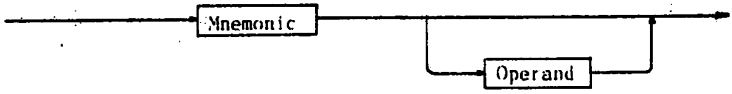
Kommentar



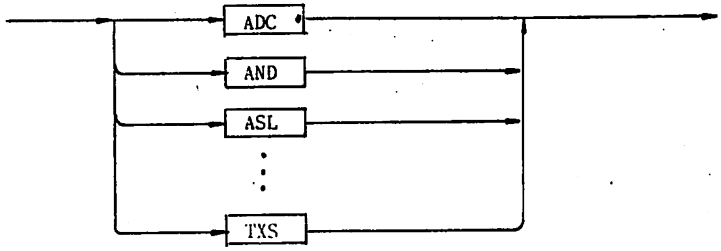
Origin



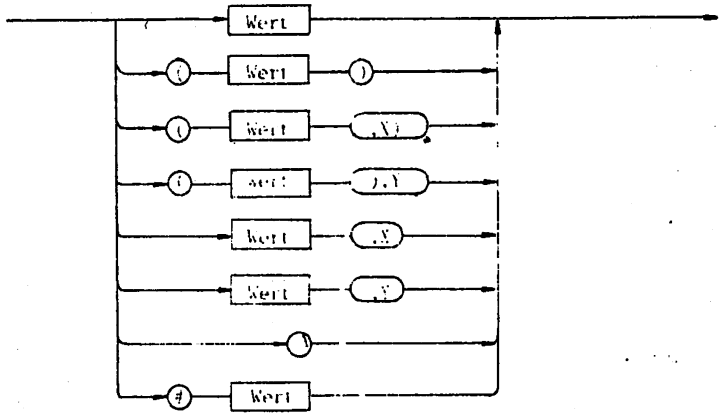
Befehl



Mnemonic

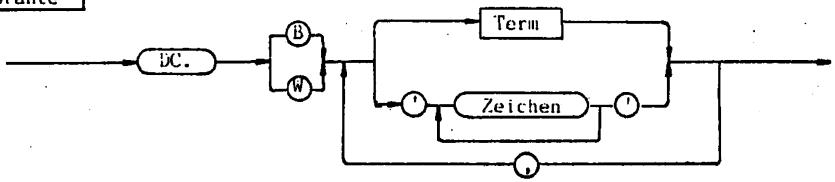


Operand

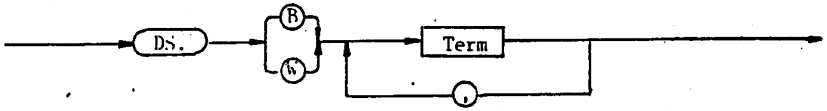


15

Konstante



Variable



2.4 Fehlermeldungen

Grundsätzlich werden alle Fehler die im Text entdeckt werden durch einen Pfeil markiert. Die Stelle des Pfeiles zeigt an, wo die Syntax des Assemblers verletzt wurde. Aufgrund der Implementierung kann maximal ein Fehler pro übersetzter Zeile angezeigt werden. (Immer nur der erste der auftrat) Insgesamt werden zwischen 9 Fehlermeldungen des Assemblers unterschieden.

2.4.1 Syntax-Fehler

Erscheint 'Syntax-Error' als Fehlermeldung, so wurde ein Zeichen erkannt, das nicht in das übliche Schema der Syntax im Assembler passt. Als Syntax-Error werden genau diejenigen Fehler deklariert, die in keine sonstige Fehler-Klasse eingeordnet werden können.

2.4.2 Byte-Groesse

'Nur Byte-Groesse erlaubt' bedeutet : Es wurde versucht ein Operand zu verwenden, dessen Wert grösser als 255 war, die geforderte Adressierungsart jedoch nur Zahlen im Bereich 0 bis 255 zulässt. Mögliche Fehlermeldung auch bei 'DC.B' mit derselben Bereichseinschränkung.

2.4.3 Wort-Groesse

'Nur Wort-Groesse erlaubt' bezeichnet eine zur obigen Fehlerart ähnliche Fehlermeldung, wobei der Zahlenbereich nun bis 65535 erweitert ist.

2.4.4 Wertebereich überschritten

Wird 'Wertebereich ueberschritten' gemeldet, so ist dies ein Anzeichen, daß versucht wurde an eine Adresse (Marke/Label) zu springen, die mehr als +127 oder -128 von der momentanen Position des Befehlszählers entfernt liegt. Mögliche Fehlermeldung auch bei Bereichsüberschreitung von EQU-Definitionen.

2.4.5 Klammer vergessen

Wurde bei einer der Indirekten Adressierungsarten eine Klammer vergessen (geschlossene Klammer), so wird dies durch diese Fehlermeldung angezeigt, das selbsständige einsetzen dieser erfolgt jedoch nicht !

2.4.6 ,X oder ,Y vergessen

Diese Fehlermeldung kann nur bei einer der Adressierungsarten ($..,X); ($..),Y; $....,X; ($....,X) oder,$....,Y auftreten. Der Sinn ist dann wohl klar.

2.4.7 Unerlaubte Adressierungsart

Wurde versucht einen Operanden in einer Adressierungsart zu adressieren, die nicht für den verwendeten Befehl vorgesehen ist, so wird 'unerlaubte Adressierungsart' als Fehlermeldung ausgegeben. Es ist dann nochmals in der Befehlsübersicht nachzusehen, welche Adressierungsarten für diesen Befehl in Frage kommen.

2.4.8 Undefiniertes Symbol

Ein Symbol ist nur dann definiert wenn es als EQU-Declariert ist oder mit einem Doppelpunkt versehen wurde. Der Wert für jedes definierte Symbol befindet sich in der Symboltabelle mit der dazugehörigen Größe. Wurde nun eine Symbol als Operand verwendet, dessen Wert nach dem 1. Assemblerlauf noch nicht festlag, so wird die Fehlermeldung 'undefiniertes Symbol' auf den Bildschirm gebracht. Ursachen hierfür können auch Schreibfehler sein, wenn ein Symbol falsch abgeschrieben wurde, so taucht es nämlich in der Symboltabelle mit dem Attribut 0005 auf. (= undefiniert)

2.4.9 Mehrfach definiert

Wie im vorigen Abschnitt gilt hier, daß ein Symbol zuerst in die Symoltabelle eingetragen werden muss, um im 2. Assemblerlauf dessen Wert als Operand eines Befehls einzutragen. Wird nun ein Symbol 2 oder mehrmals definiert, so stimmen die Werte der Definitionen nicht mehr überein. Diesen Umstand erkennt der Assembler und gibt die Fehlermeldung 'Mehrfach definiert' aus.

§3 Der Disassembler

Der Beschreibung des Disassembler sei nur ein kleiner Abschnitt gewidmet, da sich die Bedienung auf Eingabe der Anfangs und Endadresse beschränkt.

Der Disassembler dient wohl hauptsächlich dazu, einmal eingegebenen Maschinen-Code wieder 'lesbar zu machen. (Wem nutzen schon lange Zahlenkolonnen ?)
Der Vorgang des Disassemblierens kann so beschrieben werden:

- Hole den Maschinencode und schau nach was für ein Befehl dahintersteckt.
- Stelle fest welche Adressierungsart Verwendung gefunden hat.
- Wenn Operanden benötigt werden, so erhöhe den Befehlszähler und hole den Operanden aus dem Speicher.
- Gebe Befehl mit Operand in entsprechender Adressierungsart aus

Für den in diesem System implementierten Disassembler gilt nun folgendes:

- Alle Adressenangaben beziehen sich wieder auf die BASIS-Adresse des 65c02.
- Die Assembleroptionen des Grundprogrammes sind auch hier gültig, sodaß der disassemblierte Text z.B. auch auf einen Drucker ausgegeben werden kann.
- Angehalten wird der Disassembler durch CTRL-S und mit CTRL-Q wieder gestartet.
- Als Adressangaben sind Symbole zulässig. (allerdings nur die im 65c02 Assembler definierten.)
- Ist die Anfangsadresse größer als die Endadresse findet kein Aufruf des Disassemblers statt.
- nicht vorhandene OPCODES werden als '???' gedruckt

Dies wären eigentlich alle Punkte die bei der Bedienung zu beachten sind.

§4 Der Emulator

Das Kernstück dieses Programmes ist das Emulator-Modul, welches für die Abarbeitung der 65c02 Maschinenbefehle verantwortlich ist. Bei der Befehlsdecodierung und der -Abarbeitung wurde darauf geachtet, so wenige Maschinenbefehle zur Simulation eines 6502 Maschinenbefehl zu benötigen wie dies irgend möglich war. Daß dieses Emulator-Modul natürlich ein Kompromiss zwischen Benutzerfreundlichkeit und Geschwindigkeit darstellen muß, wird klar wenn man die Debug- und Test-Möglichkeiten betrachtet. Zur Realisierung des Einzelschritt-modus mussten zwei Befehle pro Zyklus zusätzlich eingebracht werden. So sollten an die Geschwindigkeit des Systems keine allzugroße Hoffnungen geknüpft werden. Im Schnitt reicht die Zeit aus, um mit einem System von ca. 0.1 MHz Taktfrequenz konkurrieren zu können, wenn der 68008 mit vollen 8 MHz und 3 Wait-Zyklen läuft.
Im Gegensatz zur Prozessorspezifikation wird bei einem ungültigen Opcode die Befehlsausführung nicht mit dem nächsten Befehl fortgesetzt, sondern der virtuelle Prozessor hält an, wobei die Stelle an der die Emulation endete sowie die gesamten Registerinhalte ausgegeben werden. So können Fehler im Programm noch erkannt und ein in ein Datenfeld laufender Prozessor verhindert werden.

Der Befehl 'BRK' (Opcode $00) dient dazu den Prozessor/Emulator anzuhalten.- Wird $00 entdeckt, so wird der aktuelle Inhalt der 6502 Register ausgegeben und der Emulator befindet sich wieder im 68000-Modus. Es erscheint die blinkende Meldung 'Processor-Break stop emulating' auf dem Bildschirm, die anzeigt, daß der Emulator auf diesen Befehl gestoßen ist.

Um Irrtümer auszuschliessen sei hier noch bemerkt, daß der Emulator nur Maschinenbefehle in Form von "Hexcodes" verarbeiten kann. Die Übersetzung in diese erfolgt entweder von Hand oder durch den Assembler !

4.1 Schnittstelle mit dem Grundprogramm

Um Programme aus dem Grundprogramm verwenden zu können, wurde eine Möglichkeit geschaffen mit diesen Daten auszutauschen. Die Schnittstelle hierfür bildet der Befehl STA ($....) (Store Ackumulator indirekt). Wird als Adresse $0000 gewählt, so wird dieser Befehl zu einem Ausnahme-Befehl :

- im Ackumulator steht die Nummer des aufzurufenden Grundprogramms. Die Nummern für jedes dieser Programme finden Sie in dem Assemblerlisting des Grundprogramms.
- Die Adressen $0000 und $0001 geben an, ab wo die Übergabe-Daten im Speicher stehen. Es gilt auch hier : LSB dann MSB der Adresse. Stehen z.B. ab $4000 die Übergabe-parameter, so muß in Adresse $0 der Wert 0 und in Adresse $1 der Wert $40 abgelegt werden.
- Als Adressen dürfen nur Geradzahlige verwendet werden, da ansonsten der 68008 versucht einen Wortzugriff auf eine ungerade Adresse aus zuführen, was in einem Adresserror endet !
- Steht die Adresse fest, ab der die Parameter stehen, so werden ab dort nacheinander folgende Register mit den Werten geladen:

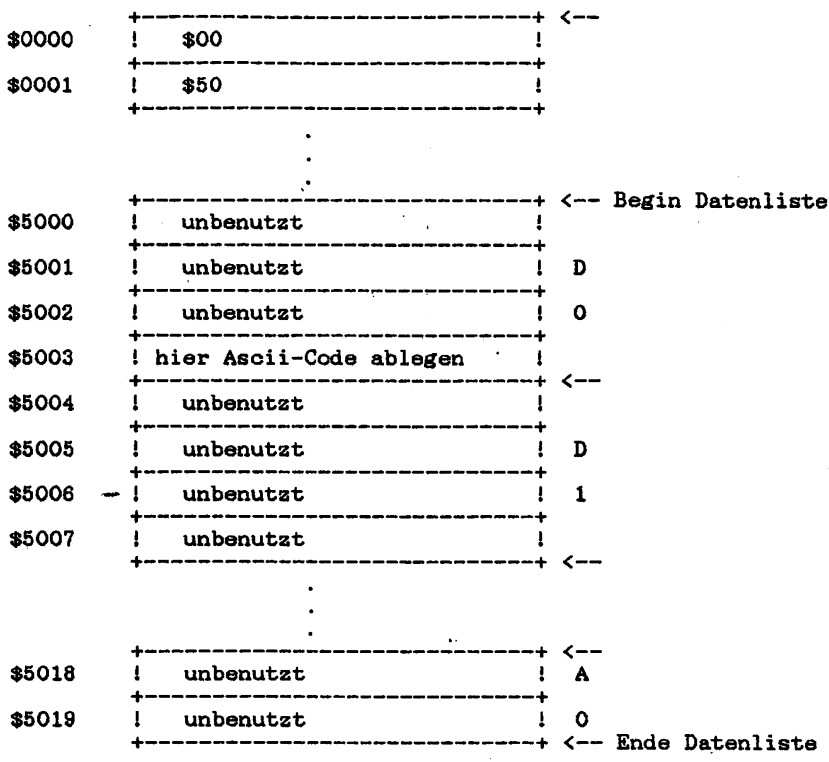
```
          ($0000) = D0.L
         4($0000) = D1.L
         8($0000) = D2.L
        12($0000) = D3.L
        16($0000) = D4.L
        20($0000) = D5.L
        24($0000) = A0.W <--- .W !!
        Wichtig !!  Hier kommen zuerst Bit 31-24 in Byte 1
                                dann   Bit 23-17 in Byte 2
                              * dann   Bit 16-8  in Byte 3
                      und schließlich Bit  7-0  in Byte 4
```

Nach Aufruf eines Grundprogramms, werden die veränderten Register wieder dorthin abgelegt.

Als Beispiel die Ausgabe eines Zeichens auf den Bildschirm :
Zeichen soll in D0 stehen, Verwendetes Grundprogramm §CO2 = Nr. 33
Daten ab $5000.

1) In Adresse $0 Wert 0 speichern.
2) in Adresse $1 Wert $50 speichern.
3) in Adresse $5003 den Ascii-Code des auszugebenden Zeichens ablegen. ($5000-$5002 werden nicht benötigt)
4) Acku mit dem Wert 33 (=$21) laden.
5) Befehl STA ($0000) ausführen. Fertig !

Graphisch:
==========


$0000
$00
$0001
$50
$5000
unbenutzt
Begin Datenliste
$5001
unbenutzt
D
$5002
unbenutzt
0
$5003
hier Ascii-Code ablegen
$5004
unbenutzt
$5005
unbenutzt
D
$5006
unbenutzt
1
$5007
unbenutzt
$5018
unbenutzt
A
$5019
unbenutzt
0
Ende Datenliste

Sollen mehrere Zeichen ausgegeben werden, so können diese z.B. mit Hilfe des X-Registers in Adresse $5003 gespeichert werden. Nachfolgender Befehl STA ($0000) ...

So ist es auf nicht allzu umständlicher Art und Weise möglich die Grundfunktionen des Rechners wie Ein- und Ausgabe, sowie die Graphikfähigkeit auch aus 65c02 Programmen mit zu benutzen.

Die Schnittstelle durch STA ($0000) stellt darüberhinaus sicher, daß nur bei dieser einen Adresse $0000 eine Ausnahme erfolgt und daß die Zykluszeit für die Emulation eines 65c02 Befehls nicht verlängert wird.
(Weitere Beispiele finden Sie im Anhang B dieses Handbuches)

Die möglichen 65c02 Befehle des Emulators entnehmen Sie bitte dem Anhang A

§5 Der Einzelschrittbetrieb

Um 65c02 Programme testen zu können, wurde noch die Möglichkeit geschaffen ein Programm in Einzelschritten auszuführen. Nach Wahl des Menu-Punktes 'Einzelschritt' kann die Startadresse, des in Einzelschritten auszuführenden Programmes, eingegeben werden. Hierfür gilt wieder die Regel, daß der Programmstart relativ zur Basis-Adresse erfolgt und die Adressen im Bereich von $0000 bis $ffff liegen müssen.

Es sollten nun sofort die 65c02 Registerinhalte auf dem Bildschirm zu sehen sein, sowie rechts-unten eine Zeile wie sie vom Disassembler erzeugt wird. Diese Zeile entspricht der DEBUG-Info des 68000 Grundprogrammes, mit dem Unterschied, daß kein zusätzliches RAM für eine solche Tabelle bereitgestellt werden muss, da die Zeile vom Disassembler erzeugt wird. Verhält sich der Prozessor nämlich anders als erwartet, so ist es durch die Ausgabe mit dem Disassembler, immer möglich festzustellen welcher Befehl als nächstes ausgeführt wird.

Die Angaben der Register bei Prozessor-Stop oder Einzelschrittbetrieb erfolgt durch die Auflistung der Registernamen mit deren Inhalt. Wobei
AC = Akku - / X = X-Register / Y = Y-Register
SP = Stackpointer / SR = Statusregister und PC = Befehlszähler

Befehlszähler und Stackpointer sind 16-Bit gross, die restlichen Register jedoch nur 8-Bit gross. Für Stackpointer und Statusregister gelten besondere Regeln : Der Stackpointer ist zwar eine 16-Bit Adresse, in Wirklichkeit ist er aber auf die Seite 1 ($0100 - $01ff) beschränkt. Das Statusregister ist 8-Bit groß, wobei im Einzelschrittbetrieb und bei Prozessor-Stop alle 8 Bit einzeln angezeigt werden, sodaß die einzelnen Bedingungen besser abgelesen werden können.

5.1 Umgang mit dem Einzelschritt-Modus des 65c02

Befindet sich der Prozessor im Einzelschrittbetrieb, so reagiert er auf bestimmte Befehle von der Tastatur, die mit einem einzigen Tastendruck aufgerufen werden. Die Fortschaltung zum nächsten auszuführenden Maschinenbefehl wird durch Drücken der RETURN-Taste erreicht. Hier nun eine Übersicht der Steuerungsmöglichkeiten, die während eines Einzelschritt-Ablaufs gültig sind :

5.1.1 B = Breakpoint

Durch Drücken von 'B' auf der Tastatur wird zur Eingabe eines Breakpointes aufgefordert. Ist die Adresse (auch Symbol) eingegeben, so wird die Ausführung des Programmes im Schnellauf fortgesetzt, bis die eingegebene Adresse im Befehlszähler erscheint. Danach wird im Einzelschritt-Modus weitergearbeitet. Wurde die eingegebene Adresse nicht erreicht, so wird das Programm bis zum "Auftauchen" eines ungültigen Opcodes oder von $00 (=BRK) ausgeführt.

5.1.2 N = Nmal

Wurde 'N' eingegeben, so erscheint eine Eingabe-Aufforderung, bei der Sie die Anzahl der "schnell" auszuführenden Befehle eingeben sollten. Der Prozessor wird daraufhin veranlasst, genau diese Anzahl auszuführen, um dann wieder in den Einzelschrittmodus zurückzukehren.

5.1.3 P = Seite

Im Gegensatz zu allen weiteren Optionen und zum Einzelschritt-Betrieb des Grundprogrammes müssen hier zwei Tastendrücke aufeinanderfolgend eingegeben werden. Erster Tastendruck gleich 'P', zweiter Tastendruck gleich Nummer der Seite die gewünscht wird (0..3). Achtung !! Es erfolgt eine Rückmeldung ob 'P' eingegeben wurde, durch Anzeige des Buchstaben 'P' rechts unten im gerade gültigen Schreibfenster.

5.1.4 C = Verändere Registerinhalte

Durch Eingabe von 'C' kann eine Veränderung der Registerinhalte erfolgen. Sie werden aufgefordert eines der Register auszuwählen, um ihm einen neuen Wert zu geben. Die Änderung erfolgt durch 'A:=Wert' wenn der Ackuinhalt, 'X:=Wert' wenn der X-Registerinhalt, 'Y:=Wert' wenn der Y-Registerinhalt und durch 'SR:=Wert' wenn der Statusregisterinhalt verändert werden soll. Als 'Wert' sind nur Byte-Größen erlaubt; werden größere Werte verwendet, so muß die Eingabe wiederholt werden.

5.1.5 T = Setze eine von 4 möglichen Triggerbedingungen

Als weitere Möglichkeit in den Programmlauf von außen einzugreifen, wurde das Setzen spezieller Triggerbedingungen vorgesehen. Das Programm wird solange ausgeführt bis diejenige Bedingung erfüllt ist, die im Einzelschrittmodus, durch drücken von 'T', definiert wurde. Möglichkeiten sind hierbei :

1. Acku als Bedingung
2. X-Register als Bedingung
3. Y-Bedingung als Bedingung
4. Eine Speicherzelle als Bedingung

Definiert wird durch : 'A=Wert' wenn 1. 'X=Wert' wenn 2. 'Y=Wert' wenn 3. und 'Madresse=Wert' wenn 4. als Bedingung gewünscht wird. Dabei ist Wert eine Bytegröße.

Beispiel hierzu: Das Programm soll solange ausgeführt werden, bis das erstemal der Wert $AA im Acku erscheint. Hierzu waehle man den Menu-Punkt 'Einzelschritt' und gebe die Startadresse ein. Nun sollte am unteren Bildschirmrand die Registerliste und der 1. auszuführende Befehl erscheinen. Drücken Sie nun auf die Taste 'T'; es erscheint jetzt die Nachricht :

' Setze Triggerbedingung ' am oberen Bildschirmrand,
' A=.. X=.. Y=.. M....=.. ' am unteren Bildschirmrand und
die Eingabeaufforderung ' Bed. + Inhalt ' in Bildschirmmitte.
Geben-Sie 'A=$AA' ein und Drücken auf die RETURN-Taste.
Das Programm wird jetzt im Schnellmodus ausgeführt, bis der Ackuinhalt das erstemal $AA ist. Die Registerliste etc. wird während des Schnellaufes nicht mehr angezeigt. Erscheint niemals der Inhalt $AA im Acku, so wird das Programm ordnungsgemäß ausgeführt bis ein ungültiger OPCODE oder ein BRK entdeckt wird. Es bestehen weiterhin alle o.g. Eingriffs-Möglichkeiten, das Programm zu unterbrechen.

5.1.6 M = Menu

Wird 'M' eingeben, so wird der Einzelschrittmodus verlassen. Es erscheint nun 'F= Flip M=Menu', um ins 65c02 Menu zurückzukehren.

5.1.7 R = Runmodus

Durch Drücken von 'R' ist es möglich, ein Programm im Schnellmodus ablaufen zu lassen. Die Registerinhalte werden dabei nicht mehr angezeigt. Das Programm wird nun in fast derselben Zeit ausgeführt, wie dies bei 'Starten' des 65c02 Menu's geschieht, mit dem Unterschied, daß der Programmablauf durch obige Optionen unterbrochen und dann wieder fortgesetzt werden kann.

5.1.8 S = Stop

'S' hat die entgegengesetzte Wirkung des 'Run-Befehls'. Wurde ein Programm im Einzelschrittbetrieb durch 'R' zur Schnellausführung gebracht, so kann es nun wieder angehalten werden. Nach Drücken der Taste 'S' befindet sich das Programm wieder im Einzelschrittbetrieb.

Soviel zum Einzelschrittbetrieb des Emulators.

Anhang A : Assembler Mnemonics des 65c02

Alphabetische Auflistung aller Assemblerbefehle des 65c02 Emulators

ADC Add memory to accumulator with carry ADC Flags : N Z C I D V
x x x - - x

Addressierungsart	Assemblerbefehl	Opcode
ö Immideate	ö ADC #Wert8	ö 69 ö
ö Zero Page	ö ADC Wert8	ö 65 ö
ö Zero Page, x	ö ADC Wert8,X	ö 75 ö
ö Absolute	ö ADC Wert16	ö 6D ö
ö Absolute, x	ö ADC Wert16,X	ö 7D ö
ö Absolute, y	ö ADC Wert16,Y	ö 79 ö
ö (Indirect,x)	ö ADC (Wert8,X)	ö 61 ö
ö (indirekt),y	ö ADC (Wert8),Y	ö 71 ö
ö Indirect	ö ADC (Wert8)	ö 72 ö

AND "AND" Memory with accumulator AND Flags : N Z C I D V
x x - - - -

Addressierungsart	Assemblerbefehl	Opcode
ö Immideate	ö AND #Wert8	ö 29 ö
ö Zero Page	ö AND Wert8	ö 25 ö
ö Zero Page, x	ö AND Wert8,X	ö 35 ö
ö Absolute	ö AND Wert16	ö 2D ö
ö Absolute, x	ö AND Wert16,X	ö 3D ö
ö Absolute, y	ö AND Wert16,Y	ö 39 ö
ö (Indirect,x)	ö AND (Wert8,X)	ö 21 ö
ö (indirekt),y	ö AND (Wert8),Y	ö 31 ö
ö Indirect	ö AND (Wert8)	ö 32 ö

ASL Shift Left One Bit (Memory or Accumulator) ASL Flags : N Z C I D V
x x x - - -

Addressierungsart	Assemblerbefehl	Opcode
ö Accumulator	ö ASL A	ö 0A ö
ö Zero Page	ö ASL Wert8	ö 06 ö
ö Zero Page, x	ö ASL Wert8,X	ö 16 ö
ö Absolute	ö ASL Wert16	ö 0E ö
ö Absolute, x	ö ASL Wert16,X	ö 1E ö

BCC Branch on Carry Clear BCC Flags : N Z C I D V
- - - - - -

Addressierungsart	Assemblerbefehl	Opcode
ö Relative	ö BCC Wert8	ö 90 ö

BCS Branch on Carry set — BCS Flags : N Z C I D V

- - - - - -

Addressierungsart	Assemblerbefehl		Opcode
ö Relative	ö BCS	Wert8	ö B0 ö

BEQ Branch on result zero — BEQ Flags : N Z C I D V

- - - - - -

Addressierungsart	Assemblerbefehl		Opcode
ö Relative	ö BEQ	Wert8	ö F0 ö

BIT Test bits in memory with accumulator — BIT Flags : N Z C I D V

x x - - - x

Addressierungsart	Assemblerbefehl		Opcode
ö Immideate	ö BIT	#Wert8	ö 89 ö
ö Zero Page	ö BIT	Wert8	ö 24 ö
ö Zero Page, x	ö BIT	Wert8,X	ö 34 ö
ö Absolute	ö BIT	Wert16	ö 2C ö
ö Absolute, x	ö BIT	Wert16,X	ö 3C ö

BMI Branch on result minus — BMI Flags : N Z C I D V

- - - - - -

Addressierungsart	Assemblerbefehl		Opcode
ö Relative	ö BMI	Wert8	ö 30 ö

BNE Branch on result not zero — BNE Flags : N Z C I D V

- - - - - -

Addressierungsart	Assemblerbefehl		Opcode
ö Relative	ö BNE	Wert8	ö D0 ö

BPL Branch on result plus — BPL Flags : N Z C I D V

- - - - - -

Addressierungsart	Assemblerbefehl		Opcode
ö Relative	ö BPL	Wert8	ö 10 ö

BRA Branch always — BRA Flags : N Z C I D V

- - - - - -

Addressierungsart	Assemblerbefehl		Opcode
ö Relative	ö BRA	Wert8	ö 80 ö

BRK Force Break BRK Flags : N Z C I D V
                              - - - 1 - -

Addressierungsart	Assemblerbefehl	Opcode
Implied	BRK	00

BVC Branch on overflow clear BVC Flags : N Z C I D V
                               - - - - - -

Addressierungsart	Assemblerbefehl	Opcode
Relative	BVC Wert8	50

BVS Branch on overflow set BVS Flags : N Z C I D V
                             - - - - - -

Addressierungsart	Assemblerbefehl	Opcode
Relative	BVS Wert8	70

CLC Clear carry flag CLC Flags : N Z C I D V
                           - - 0 - - -

Addressierungsart	Assemblerbefehl	Opcode
Implied	CLC	18

CLD Clear decimal mode CLD Flags : N Z C I D V
                                - - - - 0 -

Addressierungsart	Assemblerbefehl	Opcode
Implied	CLD	D8

CLI Clear interrupt disable bit CLI Flags : N Z C I D V
                                      - - - 0 - -

Addressierungsart	Assemblerbefehl	Opcode
Implied	CLI	58

CLV Clear overflow flag CLV Flags : N Z C I D V
                                  - - - - - 0

Addressierungsart	Assemblerbefehl	Opcode
Implied	CLV	B8

CMP Compare memory and accumulator CMP Flags : N Z C I D V
                                                x x x - - -

Addressierungsart	Assemblerbefehl		Opcode
Immideate	CMP	#Wert8	C9
Zero Page	CMP	Wert8	C5
Zero Page, x	CMP	Wert8,X	D5
Absolute	CMP	Wert16	CD
Absolute, x	CMP	Wert16,X	DD
Absolute, y	CMP	Wert16,Y	D9
(Indirect,x)	CMP	(Wert8,X)	C1
(indirekt),y	CMP	(Wert8),Y	D1
Indirect	CMP	(Wert8)	D2

CPX Compare Memory and Index X CPX Flags : N Z C I D V
x x x - - -

Addressierungsart	Assemblerbefehl	Opcode
Immideate	CPX #Wert8	E0
Zero Page	CPX Wert8	E4
Absolute	CPX Wert16	EC

CPY Compare Memory and Index X CPY Flags : N Z C I D V
x x x - - -

Addressierungsart	Assemblerbefehl	Opcode
Immideate	CPY #Wert8	C0
Zero Page	CPY Wert8	C4
Absolute	CPY Wert16	CC

DEA Decrement Accumulator by one DEA Flags : N Z C I D V
x x - - - -

Addressierungsart	Assemblerbefehl	Opcode
Implied	DEA	3A

DEX Decrement index X by one DEX Flags : N Z C I D V
x x - - - -

Addressierungsart	Assemblerbefehl	Opcode
Implied	DEX	CA

DEY Decrement index Y by one DEY Flags : N Z C I D V
x x - - - -

Addressierungsart	Assemblerbefehl	Opcode
Implied	DEY	88

EOR "Exclusive-Or" memory with accumulator EOR Flags : N Z C I D V
x x - - - -

Addressierungsart	Assemblerbefehl	Opcode
Immideate	EOR #Wert8	49
Zero Page	EOR Wert8	45
Zero Page, x	EOR Wert8,X	55
Absolute	EOR Wert16	4D
Absolute, x	EOR Wert16,X	5D
Absolute, y	EOR Wert16,Y	59
(Indirect,x)	EOR (Wert8,X)	41
(indirekt),y	EOR (Wert8),Y	51
Indirect	EOR (Wert8)	52

INA Increment Accumulator by one INA Flags : N Z C I D V
x x - - - -

Addressierungsart	Assemblerbefehl	Opcode
Implied	INA	1A

INX Increment index X by one INX Flags : N Z C I D V
x x - - - -

Addressierungsart	Assemblerbefehl	Opcode
Implied	INX	E8

INY Increment index Y by one INY Flags : N Z C I D V
x x - - - -

Addressierungsart	Assemblerbefehl	Opcode
Implied	INY	C8

JMP Jump to new location JMP Flag : N Z C I D V
- - - - - -

Addressierungsart	Assemblerbefehl	Opcode
Absolute	JMP Wert16	4C
(Indirect,x)	JMP (Wert16,X)	7C
Indirect	JMP (Wert16)	6C

JSR Jump to new location saving return adress Flag : N Z C I D V
- - - - - -

Addressierungsart	Assemblerbefehl	Opcode
Absolute	JSR Wert16	20

LDA Load accumulator with memory LDA Flags : N Z C I D V
x x - - - -

Addressierungsart	Assemblerbefehl	Opcode
Immideate	LDA #Wert8	A9
Zero Page	LDA Wert8	A5
Zero Page, x	LDA Wert8,X	B5
Absolute	LDA Wert16	AD
Absolute, x	LDA Wert16,X	BD
Absolute, y	LDA Wert16,Y	B9
(Indirect,x)	LDA (Wert8,X)	A1
(indirekt),y	LDA (Wert8),Y	B1
Indirect	LDA (Wert8)	B2

LDX Load index X with memory LDX Flags : N Z C I D V

x x - - - -

Addressierungsart	Assemblerbefehl	Opcode
Immideate	LDX #Wert8	A2
Zero Page	LDX Wert8	A6
Zero Page, y	LDX Wert8,Y	B6
Absolute	LDX Wert16	AE
Absolute, y	LDX Wert16,Y	BE

LDY Load index Y with memory LDY Flags : N Z C I D V

x x - - - -

Addressierungsart	Assemblerbefehl	Opcode
Immideate	LDY #Wert8	A0
Zero Page	LDY Wert8	A4
Zero Page, x	LDY Wert8,X	B4
Absolute	LDY Wert16	AC
Absolute, x	LDY Wert16,X	BC

LSR Shift right One Bit (Memory or Accumulator) LSR Flags : N Z C I D V

0 x x - - -

Addressierungsart	Assemblerbefehl	Opcode
Accumulator	LSR A	4A
Zero Page	LSR Wert8	46
Zero Page, x	LSR Wert8,X	56
Absolute	LSR Wert16	4E
Absolute, x	LSR Wert16,X	5E

NOP No operation NOP Flags : N Z C I D V

- - - - - -

Addressierungsart	Assemblerbefehl	Opcode
Implied	NOP	EA

ORA "Or" memory with accumulator ORA Flags : N Z C I D V

x x - - - -

Addressierungsart	Assemblerbefehl	Opcode
Immideate	ORA #Wert8	09
Zero Page	ORA Wert8	05
Zero Page, x	ORA Wert8,X	15
Absolute	ORA Wert16	0D
Absolute, x	ORA Wert16,X	1D
Absolute, y	ORA Wert16,Y	19
(Indirect,x)	ORA (Wert8,X)	01
(indirekt),y	ORA (Wert8),Y	11
Indirect	ORA (Wert8)	12

PHA Push accumulator on stack PHA Flags : N Z C I D V
- - - - - -

Addressierungsart		Assemblerbefehl		Opcode	
ö Implied	ö	PHA	ö	48	ö

PHP Push processor status on stack PHP Flags : N Z C I D V
- - - - - -

Addressierungsart		Assemblerbefehl		Opcode	
ö Implied	ö	PHP	ö	08	ö

PHX Push index X on stack PHX Flags : N Z C I D V
- - - - - -

Addressierungsart		Assemblerbefehl		Opcode	
ö Implied	ö	PHX	ö	DA	ö

PHY Push index Y on stack PHY Flags : N Z C I D V
- - - - - -

Addressierungsart		Assemblerbefehl		Opcode	
ö Implied	ö	PHY	ö	5A	ö

PLA Pull accumulator on stack PLA Flags : N Z C I D V
x x - - - -

Addressierungsart		Assemblerbefehl		Opcode	
ö Implied	ö	PLA	ö	68	ö

PLP Pull processor status on stack PLP Flags : N Z C I D V
from Stack

Addressierungsart		Assemblerbefehl		Opcode	
ö Implied	ö	PLP	ö	28	ö

PLX Pull index X on stack PLX Flags : N Z C I D V
x x - - - -

Addressierungsart		Assemblerbefehl		Opcode	
ö Implied	ö	PLX	ö	FA	ö

PLY Pull index Y on stack PLY Flags : N Z C I D V
x x - - - -

Addressierungsart	Assemblerbefehl	Opcode
Implied	PLY	7A

ROL Rotate one Bit left (Memory or Accumulator) ROL Flags : N Z C I D V
x x x - - -

Addressierungsart	Assemblerbefehl	Opcode
Accumulator	ROL A	2A
Zero Page	ROL Wert8	26
Zero Page, x	ROL Wert8,X	36
Absolute	ROL Wert16	2E
Absolute, x	ROL Wert16,X	3E

ROR Rotate one Bit right (Memory or Accumulator) ROR Flags : N Z C I D V
x x x - - -

Addressierungsart	Assemblerbefehl	Opcode
Accumulator	ROR A	6A
Zero Page	ROR Wert8	66
Zero Page, x	ROR Wert8,X	76
Absolute	ROR Wert16	6E
Absolute, x	ROR Wert16,X	7E

RTS Return from subroutine RTS Flags : N Z C I D V
- - - - - -

Addressierungsart	Assemblerbefehl	Opcode
Implied	RTS	60

SBC Subtract memory from accu with borrow SBC Flags : N Z C I D V
x x x - - x

Addressierungsart	Assemblerbefehl	Opcode
Immideate	SBC #Wert8	E9
Zero Page	SBC Wert8	E5
Zero Page, x	SBC Wert8,X	F5
Absolute	SBC Wert16	ED
Absolute, x	SBC Wert16,X	FD
Absolute, y	SBC Wert16,Y	F9
(Indirect,x)	SBC (Wert8,X)	E1
(indirekt),y	SBC (Wert8),Y	F1
Indirect	SBC (Wert8)	F2

SEC Set carry flag SEC Flags : N Z C I D V
- - 1 - - -

Addressierungsart	Assemblerbefehl	Opcode
Implied	SEC	38

SED Set decimal mode SED Flags : N Z C I D V
- - - - 1 -

Addressierungsart	Assemblerbefehl	Opcode
Implied	SED	F8

SEI Set interrupt disable status SEI Flags : N Z C I D V
- - - 1 - -

Addressierungsart	Assemblerbefehl	Opcode
Implied	SEI	78

STA Store accumulator in memory STA Flags : N Z C I D V
- - - - - -

Addressierungsart	Assemblerbefehl		Opcode
Zero Page	STA	Wert8	85
Zero Page, x	STA	Wert8,X	95
Absolute	STA	Wert16	8D
Absolute, x	STA	Wert16,X	9D
Absolute, y	STA	Wert16,Y	99
(Indirect,x)	STA	(Wert8,X)	81
(indirekt),y	STA	(Wert8),Y	91
Indirect	STA	(Wert8)	92

STX Store index X in memory STX Flags : N Z C I D V
- - - - - -

Addressierungsart	Assemblerbefehl		Opcode
Zero Page	STX	Wert8	86
Zero Page, y	STX	Wert8,Y	96
Absolute	STX	Wert16	8E

STY Store index Y in memory STY Flags : N Z C I D V
- - - - - -

Addressierungsart	Assemblerbefehl		Opcode
Zero Page	STY	Wert8	84
Zero Page, x	STY	Wert8,X	94
Absolute	STY	Wert16	8C

STZ Store Zero in memory STZ Flags : N Z C I D V
- - - - - -

Addressierungsart	Assemblerbefehl		Opcode
Zero Page	STZ	Wert8	64
Zero Page, x	STZ	Wert8,X	74
Absolute	STZ	Wert16	9C
Absolute, x	STZ	Wert16,X	9E

TAX Transfer accumulator to index X TAX Flags : N Z C I D V
x x - - - -

Addressierungsart	Assemblerbefehl	Opcode
Implied	TAX	AA

TAY Transfer accumulator to index Y TAY Flags : N Z C I D V
x x - - - -

Addressierungsart	Assemblerbefehl	Opcode
Implied	TAY	A8

TRB Test and reset memory bits with accu TRB Flags : N Z C I D V
x x - - - -

Addressierungsart	Assemblerbefehl		Opcode
Zero Page	TRB	Wert8	14
Absolute	TRB	Wert16	1C

TSB Test and set memory bits with accu TSB Flags : N Z C I D V
x x - - - -

Addressierungsart	Assemblerbefehl		Opcode
Zero Page	TSB	Wert8	04
Absolute	TSB	Wert16	0C

TSX Transfer stack pointer to index X TSX Flags : N Z C I D V
x x - - - -

Addressierungsart	Assemblerbefehl	Opcode
Implied	TSX	BA

TYA Transfer index Y to accumulator TYA Flags : N Z C I D V
                                                x x - - - -

Addressierungsart	Assemblerbefehl	Opcode
Implied	TYA	98

TXA Transfer index X to accumulator TXA Flags : N Z C I D V
                                                x x - - - -

Addressierungsart	Assemblerbefehl	Opcode
Implied	TXA	8A

TXS Transfer index X to stack pointer TXS Flags : N Z C I D V
                                                - - - - - -

Addressierungsart	Assemblerbefehl	Opcode
Implied	TXS	9A

Anhang B Programmbeispiele in 65c02 Assembler

Allgemeines:

Die unten aufgeführten Beispiele sollen die Arbeitsweise des Emulators aufzeigen. Hierzu können die Beispielprogramme ohne jede Änderung in den Texteditor eingegeben werden. Es soll nun gezeigt werden, wie die Programmentwicklung üblicherweise von Statten geht :

(i) Aufruf des Menupunktes "EDITOR" im Grundprogramm-Menu.

(ii) Text eingeben (wie unten aufgelistet). Dabei auf mögliche Fehler achten. Keine Blanks einfügen, wo sie unerlaubt sind. (Z.B. bei Bezeichnern und Befehlsmnemonics)

(iii) Texteditor verlassen und Assembleroption setzen.(je nach Wunsch die Ausgabe umlenken oder nur Fehlerausgabe veranlassen)

(iv) Wählen Sie nun den Menüpunkt "BIBLIOTHEK" aus.
Solange <cr>=weiter eingeben bis 'EMULATOR' im Bildfenster erscheint, jetzt 'J' für JA drücken.

(v) Wurde RAMSTART noch nicht gesetzt, so ist noch eine Adresse anzugeben, die der 65c02 als Adresse $0000 ansieht.

(vi) Menüpunkt "Assemblieren" anwählen.
Haben sich Fehler in ihr Programm eingeschlichen, so sollten Sie mit "W" das 65c02 Menu verlassen und den EDITOR des Grundprogrammes aufrufen. (jetzt weiter wie unter Punkt (ii) genannt)

(vii) War der übersetzungslauf erfolgreich, kann das Programm mit Hilfe des EMULATIONS-Programm abgearbeitet werden.
Nun ist es gleichgültig, ob Sie die Funktion "STARTEN" oder "EINZELSCHRITT" auswählen. Beidesmal muß die Startadresse ihres Programmes eingegeben werden.
Haben Sie den Menupunkt "STARTEN" gewählt, so wird ihr Programm als ganzes ausgeführt.
Beim Menüpunkt "EINZELSCHRITT" wird ihr Programm in einzelnen Schritten ausgeführt. (Ein wichtiges Hilfsmittel bei logischen Fehlern im Program)

(viii) Wurde Programmcode im Speicher abgelegt, so kann dieser mit Hilfe des Disassemblers noch einmal in "lesbarer" Form betrachtet werden, hierzu wird der Menupunkt "DISASSEMBLIEREN" angewählt.

Nachfolgend nun die kleinen Beispielprogramme, welche Sie nach obigem Schema eingeben und ausprobieren können.

```
*
*   Programmbeispiele in 65c02 Assembler:
*
*
*   Beispiel 1:  Maximum einer Liste von Zahlen
*                Listenstart bei $42
*                in $41 steht die Anzahl der Listenelemente
*
START:  LDX     $41             ; $41 enthaelt Anzahl der Listenelemente
        LDA     #0
MAXM:   CMP     $41,X
        BCS     KLEINER
        LDA     $41,X           ; Wert ist groesser als Ackuinhalt
KLEINER:DEX
        BNE     MAXM
        STA     $40             ; Maximum nach $40 abspeichern
        BRK
```

```
*
*   Beispiel 2: Sedezimal in Ascii umwandeln
*               Wandle den Inhalt von $40 in ein ASCII-Zeichen um.
*               die vier hoechsten Bit muessen 0 sein ($40 = %0000xxxx)
*               Das ASCII-Zeichen wird in $41 abgelegt.
ASCII:  lda     $40
        cmp     #10             ; groesser oder gleich 10 ?
        bcc     ASCZ
        adc     #6              ; A..F
ASCZ:   adc     #'0'            ; Ascii-Code '0' ist Anfangsposition
        sta     $41
        brk
*
*  Beispiel 3:  gebe einen Text auf Bildschirm aus und warte bis Taste gedrueckt
*               Die Schnittstelle mit dem Grundprogramm, soll demonstriert
*               werden.
DUMMY   EQU     $0000           ; FUER SCHNITTSTELLE
GIBAUS: LDA     #20             ; FUER TRAP #1 (20 = CLRSCREEN)
        STA     (DUMMY)         ; EGAL WO DATEN FELD, DA UNVERAENDERT
        LDX     #0              ; DATENFELD AB $2000 EINRICHTEN
        STX     $0
        LDX     #$20
        STX     $1
        LDX     #0
        LDA     #33             ; BEI TRAP  #1 (33 = CO2)
GLOOP:  LDY     TEXT,X
        BEQ     ENDE            ; Textende mit 0 markiert !
        STY     $2003           ; nur Bytegroesse <=> D0.B
        STA     (DUMMY)
        INX
        BRA     GLOOP
ENDE:   LDA     #12             ; (12 = CI)
        STA     (DUMMY)         ; Warte bis Taste gedrueckt und dann ende
        BRK
; JETZT DASSELBE MIT WRITE ALS AUSGABEPROGRAMM
;
GIBAUS2:LDA     #20
        STA     (DUMMY)
        LDX     #0
        STX     $0
        LDX     #$20            ; WIEDER AB $2000 DATENFELD
        STX     $1
        LDA     #10             ; 10 IST WRITE
        LDX     #$11            ; TEXTGROESSE
        STX     $2003           ; D0.B
        LDX     #0
        STX     $2002+4         ; D1.W
        STX     $2002+8         ; D2.W
        LDX     #20             ; X
        STX     $2003+4
        LDX     #100            ; Y
        STY     $2003+8
        LDX     #<TEXT          ; NIEDERWERTIGES BYTE VON A0
        STX     $2000+23
        LDX     #>TEXT          ; HEOHERWERTIGES BYTE VON A0
        STX     $2000+22
        STA     (DUMMY)
        BRA     ENDE            ; JETZT NOCH ZEICHEN EINGEBEN ALS ENDE

TEXT:   DC.B    ' Dies ist ein Beispiel zur Textausgabe ',0
```