COMAL-Modul 2.02
für den Commodore 128

Ergänzungsband zu COMAL für den Commodore 64

2. überarbeitete Auflage

COPYRIGHT

Nutzungsbedingungen:

UNTERSTÜTZUNG

Falls Sie Anmerkungen zu diesem **COMAL-Handbuch** oder der Programmiersprache selbst haben, wenden Sie sich bitte an Ihren **Fachhändler.**

Commodore Data A/S hat sich bemüht,

1. den Inhalt des Handbuchs korrekt darzustellen,
2. die Funktionalität der Programmiersprache gemäß ihrer Spezifikation sicherzustellen.

Von Nutzern gemeldete Fehler werden schnellstmöglich behoben. Ihr Beitrag in dieser Hinsicht wird von anderen Anwendern sehr geschätzt.

HAFTUNG

Commodore Data A/S, dessen Händler oder Zulieferer übernehmen **keine Haftung** – weder ausdrücklich noch implizit – für die **Qualität, Leistung, Wertigkeit** oder **Eignung** der hier beschriebenen Programmiersprache für spezielle Anwendungen.
Verkaufsbedingungen:

- Die Software wird **"wie gesehen"** verkauft.
- Das **Risiko** hinsichtlich Qualität und Leistung trägt der **Käufer**.
- Im Falle von **Defekten nach dem Kauf** muss der Käufer (nicht UniComal, Commodore Data A/S, Händler oder Zulieferer) alle Kosten für Reparatur, Service oder zufällige Schäden übernehmen, die durch Fehler in der Software oder der begleitenden Dokumentation entstehen.

Impressum

Titel der deutschen Ausgabe:

COMAL für den Commodore 128 – Ergänzungsband (deutsche Übersetzung)

Deutsche Übersetzung und Bearbeitung:

Diese Übersetzung wurde mit freundlicher Genehmigung von Frank Bason angefertigt.

Originaltitel:

C-128 Tillæg

1. Ausgabe, 1. Auflage, Oktober 1986
   Satz und Layout: Silkeborg Soldata, Dänemark
   Druck: Silkeborg Bogtryk A/S, Dänemark

2. Auflage, Juni 2025
Enthält Ergänzungen und Korrekturen zur ersten deutschen Ausgabe.

Verantwortlich für Inhalt und Gestaltung:

Claus Schlereth

Inhaltsverzeichnis

COMMODORE 128 COMAL

In diesem C-128-Ergänzungsband werden die neuen und erweiterten Möglichkeiten beschrieben, die sie als Nutzer des C-128-COMAL-Moduls Version 2.02 verwenden können.
Dieses Buch ist eine Ergänzung zum Handbuch COMAL für Commodore 64 und sollte hinten in den Ringordner eingeheftet werden.
Mit dem C-128-COMAL-Modul steht nicht nur eine hervorragende Programmiersprache zur Verfügung. Das Modul enthält zudem Bearbeitungsfunktionen und eine Arbeitsumgebung für Programmierer, die sich mit den besten der Welt messen kann.

VORBEREITUNG DES COMPUTERS

Wer bereits das C-64-COMAL-Modul genutzt hat, sollte den Computer und Zubehörgeräte ausschalten und das C-64-Modul durch das neue C-128-Modul ersetzen.
Für neue Commodore-128-Computer folgen Sie bitte der Anleitung im C-128-Handbuch. Stellen Sie den Computer auf, schließen Sie den Monitor und das Diskettenlaufwerk (oder den Bandrekorder) an.

DIE BEIDEN TEXTMONITORE

Wenn Sie einen Farbmonitor verwenden, achten Sie darauf, ob es sich um einen **"RGB"-Monitor** handelt (der 80 Zeichen pro Zeile anzeigen kann) oder um einen **"PAL"-Typ** (nur 40 Zeichen pro Zeile). Bei Verwendung des **Commodore-1901-Monitors** können Sie zwei Kabelpaare anschließen und mithilfe eines **Schalters links auf der Bedienungsfront des Monitors** zwischen RGB (80-Zeichen-Modus) und PAL (40-Zeichen-Modus) wechseln.

Beide Monitore sind mit Ihrem **C-128-Modul** kompatibel. Der **80-Zeichen-Modus** bietet viele Vorteile, insbesondere bei der Programmbearbeitung oder für textbasierte Anwendungen. Möchten Sie jedoch **Sprites** oder andere Farbgrafiken anzeigen, ist ein Wechsel zum **40-Zeichen-Modus** erforderlich. Beachten Sie außerdem, dass die im Handbuch COMAL für Commodore 64 beschriebenen Programme stets den 40-Zeichen-Modus verwenden.

Es ist auch möglich, ein Programm auszuführen, das **beide Monitore gleichzeitig nutzt**. Sind zwei Monitore angeschlossen, können Sie auf dem **40-Zeichen-Bildschirm** Farbgrafiken (z. B. animierte Sprites) anzeigen, während auf **dem 80-Zeichen-Bildschirm** Text ausgegeben wird.

STEUERUNG DES BILDSCHIRMMODUS

- Die Prozeduren textmode(1) und textmode(0) aus dem **Systempaket** ermöglichen den Wechsel zwischen **80- und 40-Zeichen-Modus** auf dem Bildschirm während der Programmausführung.
- Über die Tastenkombination **<ESC><X>** können Sie direkt zwischen den Modi hin- und herschalten
- Die Funktion inqsys(5) aus dem Systempaket gibt Auskunft über den **aktuellen Bildschirmmodus** (Details siehe Abschnitt **Ergänzungen zum Systempaket).**

Die **<40/80 Display>-Taste** oben auf der C-128-Tastatur dient ebenfalls zur Auswahl des **Bildschirmmodus:**

- Gedrückt beim Einschalten: Aktiviert den 80-Zeichen-Modus.
- Nicht gedrückt: Startet im 40-Zeichen-Modus.

STARTVORGANG
Nachdem der Computer mit dem Modul, allen Kabeln und dem richtigen Bildschirmmodus betriebsbereit ist, schalten Sie ihn ein. Bei einem Diskettenlaufwerk legen Sie die C-128-Demodiskette ein und geben ein:

LOAD "c-128 DEMO" <Return>

Drücken Sie anschließend RUN und <Return> (oder <F7> wie beim C64). Unterbrechen Sie das Programm mit der <RUN/STOP>-Taste und listen Sie es mit LIST (oder <F6><Return>) auf, idealerweise auf dem 80-Zeichen-Bildschirm.

C-128-BILDSCHIRMBEARBEITUNG
Haben Sie sich jemals gewünscht, Programmauflistungen sowohl **aufwärts als auch abwärts** auf dem Bildschirm scrollen zu können? Mit dem neuen **C-128-COMAL-Modul** wird dieser Wunsch nun erfüllt:

- Drücken Sie **<Alt-Cursor Hoch>** (halten Sie die **<ALT>-Taste** oben auf der Tastatur – dritte Taste von links – gedrückt und betätigen Sie gleichzeitig die **Aufwärtspfeiltaste** rechts oben). Voilà! Neue Programmzeilen erscheinen von oben nach unten.
- Drücken Sie **<Alt-Cursor Runter>**, um die Liste nach oben zu scrollen.
- Auch die **Cursor-Tasten unten** auf der Tastatur können für diese Funktion genutzt werden.

Wie beim **C-64-COMAL-Modul** lässt sich die Ausgabe mit der **<Leertaste>** pausieren. Im Gegensatz zum C64-Modul kann die **<Strg>-Taste** jedoch nicht mehr zum langsamen Scrollen gedrückt gehalten werden. Verwenden Sie stattdessen **<Alt>** mit den **Cursor-Tasten**, um langsam durch die Auflistung zu navigieren.

ÄNDERUNGEN ZUM C-64-MODUL
Mit dem neuen **COMAL-Modul** hat UniComal eine **Standardisierung** der Bildschirmbearbeitung entsprechend der C-128-Architektur umgesetzt. Dadurch ergeben sich einige Unterschiede zur C-64-Version. Eine vollständige Übersicht aller **<Alt>-**, **<F4>-** und **<ESC>-Tastenkombinationen**, sowie der Anpassungen finden Sie in **Anhang B**.

NEUE MÖGLICHKEITEN MIT C-128 COMAL
Die vielfältigen Funktionen von C-128 COMAL erschließen sich am besten durch praktische Programmieraufgaben. Die kurzen Codebeispiele zu den einzelnen Befehlen sowie die umfangreicheren Beispiele auf der **C-128-Demodiskette** liefern Inspiration. Neben den offensichtlichen Verbesserungen – wie **40 KB Speicher** für Benutzerprogramme (statt 30 KB), Vollzugriff auf das C-128-Tastaturlayout und deutlich schnellere Diskettenoperationen – bietet das Modul weitere neue Funktionen. Hier die Highlights:

- **Erweiterte INPUT-Funktionen**
  - Programmierer können nun neue Prozeduren aus dem **Systempaket** nutzen, um Programme benutzerfreundlicher zu gestalten. Bei der Eingabe lassen sich Tasten wie **<Cursor hoch>** und **<Cursor runter>** verwenden, um zwischen Eingabefeldern zu springen. Details finden Sie unter:

    - inputpos (Positionierung des Eingabecursors),
    - termchars (Terminatorzeichen),
    - termpos (Terminatorposition),
    - termchar$ (Terminatorzeichen als String).

- **C-128-Schriftartenpaket**
  - Da der C-128 sowoh **40- als auch 80-Zeichen-Monitore** unterstützt, wurde das Schriftartenpaket erweitert. Nutzerdefinierte Zeichensätze sind nun auch im **80-Zeichen-Bildschirmmodus** möglich (siehe Abschnitt *Weitere Schlüsselwörter*).

- **Neue Bearbeitungsfunktionen**
  - **Scrollen**: Programmauflistungen können nun aufwärts *und* abwärts gescrollt werden.
  - **Einfügen von Zeilen**: Drücken Sie **<Alt-N>**, um automatisch nummerierte Zeilen nach der aktuellen Cursorposition einzufügen.
  - **Fenstertechnik** und Wechsel zwischen 40-/80-Zeichen-Modi. Eine vollständige Übersicht aller Bearbeitungstasten (inkl. **<Alt>**, **<F4>** und **<ESC>**) finden Sie in **Anhang B**.

- **RAM-Dateien**
  - Eine bahnbrechende Neuerung des C-128-Moduls ist die Möglichkeit, Dateien mit bis zu **40 KB direkt im RAM** zu speichern. UniComal nutzt dafür einen bisher ungenutzten Speicherbereich und stellt über 20 Prozeduren/Funktionen im **ramfiles**-Paket bereit. Eine detaillierte Beschreibung finden Sie im Abschnitt **Das neue ramfiles-Paket** sowie im Demo-Programm **"RAMFILES"** auf der C-128-Demodiskette.

- **Kommunikationsverbesserungen**
  - Die Integration zusätzlicher Prozeduren für die **RS-232-Schnittstelle** im Systempaket vereinfacht die Entwicklung benutzerfreundlicher Kommunikationsprogramme – sowohl für den Austausch mit Datenbanken als auch mit anderen Computern.

KOMPATIBILITÄT MIT C-64 COMAL

Bei der Entwicklung von C-128 COMAL hat UniComal darauf geachtet, die Migration von C-64-COMAL-Programmen zur neuen Version zu vereinfachen. (Da C-128 COMAL jedoch viele neue Prozeduren und Funktionen enthält, ist der umgekehrte Weg – also die Nutzung von C-128-Programmen auf dem C-64 – in der Regel nicht möglich.)

o C-64-PROGRAMME AUF DEM C-128 NUTZEN

Wenn Sie ein C-64-Programm mit dem neuen Modul verwenden möchten, können Sie es in der Regel wie gewohnt mit dem Befehl **LOAD** laden.
Handelt es sich jedoch um ein Programm, das mit einem Maschinencode-Programm über den LINK-Befehl verknüpft ist, muss folgende Vorgehensweise angewendet werden:

- Starten Sie den C-128 mit einem eingestecktem C64-, oder C-128-Modul.
  Laden Sie das C-64-Programm mit dem Befehl LOAD, und LISTen Sie das Programm anschließend auf die Diskette:

  LIST "Programmname"

- Stellen Sie nun sicher, dass das neue C-128-COMAL-Modul eingesetzt ist.
  Es empfiehlt sich, eine komplett neue Diskette mit dem Befehl PASS zu initialisieren, bevor Sie fortfahren. Sie können beispielsweise Folgendes schreiben:

  PASS "n0:C-128 Programm,01

  Beachten Sie, dass dies 1328 Blöcke (je 254 Zeichen) ermöglicht, wenn Sie ein 1571-Diskettenlaufwerk angeschlossen haben. Das bedeutet, dass doppelt so viele Blöcke Platz haben wie auf den 1541-Disketten des C-64.

- Legen Sie die C-64-Diskette mit dem zuvor gelisteten Programm ein ein und laden Sie das C-64-Programm mit dem Befehl ENTER:

  ENTER "Programmname"

- Nutzen Sie die Bearbeitungsfunktionen des C-128, um das Programm so anzupassen, dass es korrekt auf den 80-Zeichen-Bildschirm usw. passt. Wenn Sie fertig sind, können Sie es mit dem Befehl SAVE auf der C-128-Diskette speichern.

- Verwenden Sie anschließend mit Ihrem Assembler "C128symb" auf der C-128-Demodiskette anstelle von "C64symb", um das Maschinencode-Programm an den 128 anzupassen. Weitere Informationen finden Sie im Handbuch zum Maschinencode. Weitere Informationen hierzu finden Sie in Anhang C.
  Sobald das COMAL-Programm und der Maschinencode in Ordnung sind, können Sie das Maschinencode-Programm mit LINK verknüpfen, sodass es mit SAVE zusammen mit dem Programm in einer einzigen Datei gespeichert werden kann.

NEUE UND ÜBERARBEITETE SCHLÜSSELWÖRTER

In diesem Abschnitt finden Sie eine kurze Beschreibung aller neuen oder überarbeiteten Schlüsselwörter in C-128 COMAL. Weitere Funktionen sind bereits in COMAL für Commodore 64 beschrieben. Hier sind alle neuen/überarbeiteten Schlüsselwörter (a=Anweisung, k=Kommando, p=Prozedur, f=Funktion) thematisch gruppiert. Es wird außerdem angegeben, ob das Schlüsselwort zum COMAL-Kern gehört, eine Erweiterung darstellt oder zu einem Paket gehört.

In den folgenden Abschnitten "Ergänzung zum Systempaket", "Das neue Ramfiles-Paket" und "Weitere Schlüsselwörter" finden Sie eine detailliertere Beschreibung der einzelnen Schlüsselwörter, alphabetisch geordnet innerhalb der einzelnen Abschnitte.

Neue Optionen für PRINT- und INPUT-Anweisungen

PRINT	a,k	kernal	die Trennzeichen ; und , erhalten eine neue Bedeutung.
ZONE	a,k	kernal	funktioniert nun mit **;** statt mit **,**.
option(1)	p	system	ermöglicht die Verwendung der alten Definition von ZONE.
INPUT	a,k	kernal	; und , unterdrücken Zeilenumbrüche am Ende einer INPUT-Anweisung
Inputpos	p	system	positioniert den Cursor im INPUT-Feld.
termchars	p	system	bestimmt das/die Abschlusszeichen in der INPUT-Anweisung.
termpos	f	system	meldet die Cursorposition bei INPUT-Abschluss.
termchar$	f	system	gibt den ASCII-Code für das INPUT-Abschlusszeichen zurück.
GET$	f	kernal	liest eine bestimmte Anzahl Zeichen aus einer Datei (verfügbar sowohl im C-64- als auch im C-128-Modul).

RAM-Dateien im 40-KB-Zusatzspeicher des C-128

ramfiles	Paket	ramfiles	neues Paket für RAM-Dateien im 40-KB-Speicherbereich.
deleteallrf	p	ramfiles	löscht alle RAM-Dateien.
deleterf	p	ramfiles	löscht eine spezifische RAM-Datei.
createrf	p	ramfiles	erstellt eine RAM-Datei.
eofrf	f	ramfiles	gibt das Ende einer RAM-Datei an.
posrf	p	ramfiles	positioniert den Dateizeiger innerhalb einer RAM-Datei.
writenum	p	ramfiles	schreibt eine Zahl in eine RAM-Datei.
readnum	p	ramfiles	liest eine Zahl aus einer RAM-Datei als REF-Parameter.
getnum	f	ramfiles	gibt eine Zahl aus einer RAM-Datei zurück.
writestr	p	ramfiles	schreibt einen String in eine RAM-Datei.

writerefstr	p	ramfiles	speichert einen REF-String in einer RAM-Datei.
readstr	p	ramfiles	liest einen String aus einer RAM-Datei als REF-Parameter.
writerec	p	ramfiles	schreibt einen formatierten Datensatz in eine RAM-Datei.
writerefrec	p	ramfiles	schreibt einen formatierten REF-Datensatz in eine RAM-Datei.
readrec	p	ramfiles	liest einen formatierten Datensatz aus einer RAM-Datei.
inqrf	f	ramfiles	ermittelt das Format unbekannter RAM-Dateien.
saverf	p	ramfiles	speichert eine RAM-Datei auf Diskette.
loadrf	p	ramfiles	lädt eine RAM-Datei von der Diskette.
saveallrf	p	ramfiles	speichert alle RAM-Dateien auf Diskette.
loadallrf	p	ramfiles	lädt alle RAM-Dateien von der Diskette.
fieldtype$	f	ramfiles	gibt die Feldbeschreibung einer RAM-Datei als Text zurück.
freerf	f	ramfiles	gibt die Anzahl der freien Bytes im RAM-Dateibereich zurück.
SIZE	k	kernal	erweitert: Zeigt auch genutzte und freie Blöcke im RAM-Dateibereich an.

Zeichenkonvertierung und Kommunikation

setmapping	p	system	steuert die Zeichenkonvertierung im Dateisystem, wenn /a+ verwendet wird.
resetmapping	p	system	setzt die Zeichenzuordnung auf Standard zurück.
char'in'buffer	f	system	gibt die Anzahl der Zeichen im RS-232-Sende- oder Empfangspuffer zurück.
clearbuffer	p	system	Löscht den RS-232-Sende- oder Empfangsbuffer.
rs232status	f	system	Gibt den Wert des Statusregisters der seriellen Schnittstelle zurück.
GET$	f	kernal	Erweitert: Ermöglicht verbesserte serielle Kommunikation.
IN	a, k	kernal	IN "String$" gibt nun FALSE (d. h. 0) zurück.
option(2)	p	system	wird bei relativen Dateien verwendet, um Fehler in 1541/1571-Laufwerken zu vermeiden.
setserialport	p	system	konfiguriert die Kommunikationsparameter der seriellen Schnittstelle.
bin$	f	system	gibt einen String mit der Binärdarstellung einer Zahl zurück.
hex$	f	system	gibt einen String mit der Hexadezimaldarstellung einer Zahl zurück.

Verwendung von Schriftarten

discardfont	p	font	stellt den Standardzeichensatz wieder her.
selectfont	p	font	wechselt zum angegebenen Zeichensatz.

Bildschirmsteuerung und Farben

textmode	p	system	wechselt zwischen 40- und 80-Zeichen-Bildschirmmodus.
textwindow	p	system	definiert ein Textfenster auf dem Bildschirm.
getscreen2	p	system	kopiert den Inhalt eines Textfensters in einen String.
setscreen2	p	system	überträgt Informationen aus einem String in ein Textfenster.
textcolor	p	graphics	ändert die Cursorfarbe im 40-Zeichen-Modus.
textborder	p	graphics	ändert die Randfarbe im 40-Zeichen-Modus.
textbackground	p	graphics	ändert die Hintergrundfarbe im 40-Zeichen-Modus.
textcolors	p	system	ändert die Farbpalette des aktuellen Bildschirms.
plottext	p	graphics	schreibt Text nur auf dem 40-Zeichen-Bildschirm (nicht 80-Zeichen).
vdcpoke	p	system	schreibt direkt in die Register des VDC-Chip (80-Zeichen-Modus).
vdcpeek	f	system	liest Register des VDC-Chip (80-Zeichen-Modus).
inqsys	f	system	gibt Systemparameter wie Bildschirmmodus oder Tastaturstatus zurück.
cursormode	p	system	steuert die Cursoranzeige im 80-Zeichen-Modus (ein/aus/form).

Systemebefehle und Erweiterungen

BASIC	c	erweiterung	wechselt in den Commodore-BASIC-7.0-Modus.
monitor	p	system	startet den Maschinensprache-Monitor
cpuspeed	p	system	wählt die CPU-Taktfrequenz (1 oder 2 MHz).
TRACE	k	kernal	startet eine Programmablaufverfolgung.
bell	p	system	gibt einen abbrechbaren Signalton aus.
getshape$	f	sprite	gibt einen String mit der Sprite-Grafikdefinition zurück.
hardcopymode	p	system	steuert die Druckerausgabe des Bildschirminhalts (Hardcopy).

ERGÄNZUNGEN ZUM SYSTEMPAKET

Wenn Sie USE system als Befehl direkt über die Tastatur eingeben oder die USE system-Anweisung in einem Programm ausführen, stehen Ihnen zahlreiche nützliche Prozeduren und Funktionen zur Verfügung. Auch im C-128-Modul haben Sie Zugriff auf alle in COMAL für den Commodore 64, Kapitel 5, beschriebenen Funktionen.
Zusätzlich zu all diesen Funktionen wurden zahlreiche neue und erweiterte Optionen hinzugefügt. Diese lassen sich grob wie folgt unterteilen:

- Verbesserte Nutzung der RS-232-Schnittstelle, einschließlich der Möglichkeit, erweiterte Kommunikationsprogramme zu verwenden, sodass Ihr C-128 mit Datenbanken verwendet werden kann;

- Vollständige Nutzung des 80-Zeichen-Bildschirms des C-128, einschließlich der Verwendung von Fenstern, der Steuerung des Cursormodus, der Ausgabe von Bildschirmdaten auf den Drucker, des Ausdrucks von Anführungszeichen auf Commodore-Druckern usw.;

- Spezielle Funktionen und Prozeduren zur Herstellung der Kompatibilität mit C-64-Programmen (siehe Option).

Hier finden Sie eine alphabetisch geordnete Übersicht über die neuen und geänderten Prozeduren und Funktionen Ihres COMAL-Moduls Version 2.02.

bell(dauer#)

ist eine Prozedur zur Erzeugung des COMAL-Tonsignals. Eine Dauer von # = 1 entspricht dem normalen COMALTontsignal.
Neu ist, dass der Ton ein- und ausgeschaltet werden kann.
Beispiel 1: Geben Sie die Sequenz <Esc><H> ein – die Glocke ertönt nun nicht mehr, wenn Sie bell als Prozedur verwenden. Geben Sie die Sequenz <Esc><G> ein – die Glocke wird wieder eingeschaltet.

bin$(wert#)

(ist eine String-Funktion, die die Zeichenfolge zurückgibt, die der binären Darstellung des Parameterwerts # (0 bis 65536) entspricht. Beispielsweise gibt bin$(255) die Zeichenfolge „0000000011111111“ zurück.

char'in'buffer(richtung#)

Diese Funktion gibt die Anzahl der Zeichen im RS-232-Buffer zurück. Der Parameter richtung# gleich 0 bedeutet Empfangspuffer, und gleich 1 bedeutet Sendepuffer.

clearbuffer(richtung#)

Diese Prozedur löscht den RS-232-Buffer. Parameter siehe **‚char'in'buffer'**.

cpuspeed(speed)

Diese Prozedur legt die CPU-Taktfrequenz fest. Der Wert kann gleich 0, 1 oder 2 sein. Der Standardwert ist 0.

Beispiel:	cpuspeed(0)	COMAL wählt die Taktfrequenz selbst: Für einen Bildschirm oder eine Grafik mit 40 Zeichen wird 1 MHz gewählt, andernfalls werden 2 MHz gewählt.
	cpuspeed(1)	Erzwingt 1 MHz (konstant).
	cpuspeed(1)	Erzwingt 2 MHz (deaktiviert den 40-Zeichen-Bildschirm).

cursormode(mode#)

ist eine Prozedur, mit der die Cursordarstellung auf der 80-Zeichen-Anzeige ein- und ausgeschaltet sowie geändert werden kann. Der Parameter mode# hat folgende Bedeutung:

Mode:
0: Cursor ausschalten
1: Cursor im aktuellen Zustand einschalten
2: Nicht blinkenden Blockcursor einschalten
3: Blinkenden Blockcursor einschalten
4: Schnell blinkenden Blockcursor einschalten
5: Nicht blinkenden Liniencursor einschalten
6: Blinkenden Liniencursor einschalten
7: Schnell blinkenden Liniencursor einschalten

Beispiel:

```
0010 PAGE
0020 USE system
0030 cursormode(5)
0040 LOOP
0050 PRINT KEY$,
0060 ENDLOOP
```

Das obige Beispiel zeigt eine nicht blinkende Zeilenmarkierung auf dem Bildschirm. Tastatureingaben werden (ohne Zeilenumbrüche) ausgegeben, bis das Programm mit <Run/Stop> unterbrochen wird.

Bemerkung: Bei einer INPUT-Anweisung ist der zuletzt gewählte Cursor immer sichtbar.

Im Direktmodus sind nur Mode 2,3 und 5,6 von Bedeutung, Mode 2,3 aktiviert den Blockcursor, und Mode 5,6 den Strichcursor

getscreen2(screen$)

ist eine Prozedur, die einen Teil (oder den gesamten) Textbildschirm speichert. Die Bildinformationen zum aktuellen Fenster werden in der Zeichenfolge screen$ gespeichert. Das gespeicherte Bild kann mit der Prozedur **setscreen2**(screen$) wieder abgerufen werden. Beachten Sie, dass **getscreen2** und **setscreen2** nicht mit getscreen und setscreen kompatibel sind, da diese nur auf dem 40-Zeichen-Bildschirm und immer mit dem gesamten Bildschirm arbeiten.

Beachten Sie, dass **setscreen2** und **getscreen2** nicht verwendet werden können, um Bildschirminhalte vom 40-Zeichen-Bildschirm auf den 80-Zeichen-Bildschirm oder umgekehrt zu übertragen. Ein Beispiel für die Verwendung von **getscreen2** finden Sie auf der C-128-Demodiskette (siehe „demo.window.80“ und „demo.window.40“).

hardcopymode(charsetmode#,quotemode#)

ist eine Prozedur zur Steuerung der Ausgabe von Textbildschirmen auf einem Drucker. Der Parameter **charsetmode#** kann den Wert 0 oder 1 annehmen:

charsetmode#=0 ist der Standardwert und bedeutet, dass COMAL je nach aktueller Bildschirmanzeige Groß- und Kleinbuchstaben oder Großbuchstaben und Grafiken auswählt. Im 40-Zeichen-Modus gilt der gleiche Zeichensatz für den gesamten Bildschirm, während im 80-Zeichen-Modus beide Zeichensätze gemischt werden können.

charsetmode#=1 bedeutet, dass der Zeichensatz beim Wechsel des Bildschirms nicht geändert wird, sondern nur basierend auf der Prozedur **setprinter** im Systempaket.

Der Parameter quotemode# gibt an, wie das Zeichen " auf dem Drucker behandelt wird. Bei Commodore-Druckern führt das Schreiben einer ungeraden Anzahl von " dazu, dass der Drucker in den „Quote-Modus“ wechselt. In diesem Modus werden Steuerzeichen in negativem Text geschrieben. Dies verhindert, dass der Zeichensatz und der negative Textmodus zusammen gewechselt werden. In C-64 COMAL wurde dies durch die Umwandlung von Zeichen, die in negativem Text auf dem Bildschirm nach einer ungeraden Anzahl von " geschrieben wurden, in Steuerbefehle erreicht, die in negativem Text geschrieben werden. In C-128 COMAL gibt es erweiterte Optionen zur Behandlung dieses Problems. Diese Optionen können mit dem Parameter quotemode# eingestellt werden.

quotemode#=0 bedeutet, dass das Zeichen " als Grafik geschrieben wird, wodurch der „Quote-Modus“ im Drucker nicht aktiviert wird. Dies setzt jedoch voraus, dass der Drucker mit Commodores MPS-801 Drucker für Grafikdruck kompatibel ist. Diese Einstellung wird beim Start gewählt.

quotemode#=1 aktiviert den C-64 COMAL-Modus. In diesem Modus wird quotemode# auf 1 gesetzt, da der Wechsel des Zeichensatzes sonst zu unerwünschten Zeichen im negativen Text führen könnte.

quotemode#=2-255 bedeutet, dass anstelle des Zeichens " der Wert **CHR$(quotemode#)** geschrieben wird. Auf diese Weise ist es möglich, ein anderes Zeichen anstelle von " zu drucken, z. B. ein Apostroph ' oder **CHR$(254)**, das ein programmierbares Zeichen auf einem Commodore MPS-802 Drucker darstellt.
Wenn der „Quote-Modus“ nicht aktiviert oder vom Drucker nicht unterstützt wird, wird **quotemode#=34** gewählt, was zu einem normalen " führt.

Beim Start wird automatisch **hardcopymode(0, 0)** ausgeführt.

hex$(wert#)

ist eine String-Funktion, die den entsprechenden String für die hexadezimale Darstellung des Parameters wert# (von 0 bis 65536) zurückgibt. Zum Beispiel gibt **hex$(255)** den String "00ff" zurück.

inputpos(position#)

ist eine **Prozedur**, die die relative Startposition des Cursors in einem Eingabefeld setzt. Eine 1 bedeutet die erste Position. **inputpos** gilt für die nachfolgende INPUT-Anweisung und wird danach wieder auf 1 gesetzt. Es wird ebenfalls auf 1 zurückgesetzt, wenn RUN ausgeführt wird. Siehe das Programm „demo.termchar.80“ auf der C-128-Demodiskette.

inqsys(typ#)

ist eine Funktion, die je nach dem Parameter typ# folgende Werte zurückgibt:

typ#:
- 1: Mindestzeilenhöhe des Textfensters
- 2: Maximalzeilenhöhe des Textfensters
- 3: Mindestspaltenhöhe des Textfensters
- 4: Maximalspaltenhöhe des Textfensters
- 5: Textmodus – gibt 0 für den 40-Zeichen-Bildschirm und 1 für den 80-Zeichen-Bildschirm zurück
- 6: Aktuelle CPU-Geschwindigkeit – gibt 1 für 1 MHz und 2 für 2 MHz zurück

monitor

ist eine **Prozedur**, die verwendet werden kann, um den Maschinensprache-Monitor von COMAL aus aufzurufen. Es wird mit einem X-Befehl zurückgekehrt. Die Rückkehr erfolgt normal mit einer Fehlermeldung „monitor terminated“. Es ist daher auch möglich, den Monitor aus einem laufenden Programm aufzurufen und die Rückkehr vom Monitor durch Fehlerbehandlung zu erfassen. Dies kann bei der Fehlerbehebung von Maschinenspracheprogrammen sehr hilfreich sein. Beachte, dass der Monitor nicht aufgerufen werden kann, wenn ein „soft-geladener“ Zeichensatz im 40-Zeichen-Modus verwendet wird. Dasselbe gilt für den Split-Screen-Modus. Der Grund dafür ist, dass der Bildschirm-Editor, der aus dem Monitor aufgerufen wird, nicht auf den Speicherbereich zugreifen kann, der unter dem Monitor-ROM liegt. Ein Fehler wird angezeigt, wenn versucht wird, dies zu tun.

option(1,alt#)

ist eine **Prozedur**, mit der das alte Verhalten der Trennzeichen für PRINT, nämlich , und ;, wiederhergestellt werden kann (siehe auch ZONE). Wenn alt# TRUE ist, funktioniert PRINT nach der alten Definition, wobei der Standardwert für ZONE auf 0 gesetzt wird und das Komma als Trennzeichen für Tabulatoren verwendet wird. Semikolon bedeutet dann, dass ein einziges Leerzeichen eingefügt wird. Beim Start der Maschine und bei RUN wird automatisch **option(1, FALSE)** ausgeführt. Wenn option in einem Programm verwendet wird, das sowohl unter COMAL 2.01 als auch 2.02 ausgeführt werden soll, muss option nur dann aufgerufen werden, wenn ZONE beim Programmstart auf 1 gesetzt ist.

Beispiel:
```
0010 IF ZONE THEN option(1,1)
0020 ...
```

Es ist vorteilhaft, die neue ZONE-Definition zu verwenden, da dies unerwünschte Bedeutungen des Kommas in ASCII-Dateioperationen verhindert.

option(2,postmode#)

ist eine Prozedur, die aufgrund von Fehlern in den 1541/1571 Diskettenlaufwerken hinzugefügt wurde. (Diese Prozedur ersetzt die frühere **setrecord-delay**). Der Parameter postmode# hat folgende Bedeutung:

postmode#	bit 7	(Wert 128): Gibt an, dass der Status des Diskettenlaufwerks nach allen Positionierungen gelesen wird.
	bit 6	(Wert 64): Gibt an, dass auch nach allen Lese-/Schreiboperationen positioniert wird.
	bit 0	(Wert 1) gibt an, dass vor einer Positionierung überprüft wird, ob die vorherige Diskoperation abgeschlossen ist. Dies geschieht durch das Lesen im Diskettenlaufwerkspeicher. Diese Operation kann mit anderen Laufwerken als den 1541 oder 1571 inkompatibel sein.

Beim Einschalten des Computers wird automatisch **option(2, 128)** ausgeführt. Bis die Fehler in den 1541/1571 Diskettenlaufwerken behoben sind, ist es daher notwendig, option**(2, 128+64+1)** auszuführen. Dies macht die Diskoperationen langsamer, aber fehlerfrei. Wenn nur von einer relativen Datei gelesen wird, reicht es aus, option**(2, 128+64)** auszuführen.

resetmapping

ist eine Prozedur, die die „normale“ Zeichenabbildung (Mapping) zurücksetzt, also wie im C-64-Modul:

Normale Abbildung bei Empfang:

97 … 125	→	65 … 93	(Kleinbuchstaben)
65 … 93	→	193 … 221	(Großbuchstaben)
95	→	164	(Unterstrich)

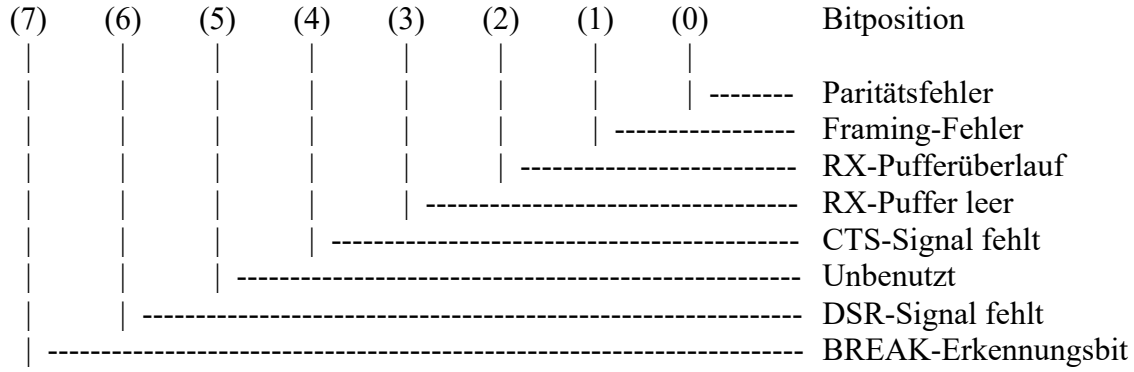
Normale Abbildung beim Senden:

65 … 93	→	97 … 125	(Kleinbuchstaben)
193 … 221	→	65 … 93	(Großbuchstaben)
164	→	95	(Unterstrich)

Beachte, dass, wenn die Abbildung geändert wird, sie für alle Dateien gilt, die mit /a+ geöffnet wurden, und dass die Abbildung so lange gilt, bis der Computer ausgeschaltet wird oder sie mit setmapping oder resetmapping geändert wird.

rs232status

ist eine Funktion, die den Wert des Statusregisters des seriellen Ports zurückgibt und das Register anschließend zurücksetzt. Die Funktion gibt folgende Werte zurück:



Beachten Sie, dass Bit [3] (RX-Puffer leer) nach GET$ abgelesen werden kann. Es wird auf 1 gesetzt, wenn die erwartete Anzahl von Zeichen nicht im RX-Puffer vorhanden war. RX ist die Abkürzung des englischen Wortes “receive”, was “empfangen” bedeutet.

setmapping(richtung#,char$,zu_char$)

ist eine Prozedur, die die Umwandlung von Zeichen bei der Kommunikation mit anderen Systemen, z.B. über eine RS-232-Schnittstelle, erleichtert. Eine Umwandlung war bereits im C-64 COMAL im Dateisystem möglich, z.B. durch das Anhängen von **/a+** an einen Dateinamen. **setmapping** ist eine Erweiterung dieser Funktion. Der Parameter richtung# ist 0 für Empfang (READ, GET oder INPUT FILE) und 1 für Senden (WRITE oder PRINT FILE). Die Prozedur **setmapping** spezifiziert, dass das Zeichen char$ durch das Zeichen zu_char$ bei nachfolgenden Übertragungen ersetzt wird.

setscreen(screen$)

ist eine Prozedur, die einen Teil des Textbildschirms überträgt, der zuvor mit **getscreen2** gespeichert wurde. Die Bildinformationen sind im String skarnz$ gespeichert. Das Bild wird in der oberen linken Ecke des aktuellen Textfensters platziert und bei Bedarf beschnitten. Außerdem werden die Markierungsfarbe und die Position gemäß den Angaben in screen$ gesetzt. Beachten Sie, dass **getscreen2** und **setscreen2** nicht mit **getscreen** und **setscreen** kompatibel sind, die nur auf dem 40-Zeichen-Bildschirm arbeiten und immer den gesamten Bildschirm betreffen. Beachten Sie auch, dass **setscreen2** und **getscreen2** nicht verwendet werden können, um Bildschirmbilder vom 40-Zeichen-Bildschirm auf den 80-Zeichen-Bildschirm oder umgekehrt zu übertragen. Ein Beispiel für die Verwendung von **getscreen2** kann in den Programmen “demo.window.40” und “demo.window.80” auf der C-128-Demodiskette gefunden werden.

setserialport(attribute$)

ist eine Prozedur, die den Standardzustand des seriellen Ports ändert (ähnlich wie setprinter das für Drucker tut). Der Parameter attribute$ ist ein String, der Informationen über die Kommunikationsparameter des seriellen Ports enthält.
Diese Spezifikationen können auch im Zusammenhang mit einer OPEN FILE-Anweisung oder -Kommando angegeben werden.
Neben der Übertragungsgeschwindigkeit, der Anzahl der Datenbits, der Stoppbits und der Parität — wie bereits in C-64 COMAL möglich — kann man jetzt außerdem Halb- oder Vollduplex, Handshake und zusätzliche Paritätsparameter festlegen.

Alle Möglichkeiten sind in der folgenden Tabelle dargestellt:

Parameter	Syntax	mögliche Werte	Standardwert
baudrate	b<baud>	50-2400	b300
databits	d<anzahl>	5-8	d7
stopbits	s<anzahl>	0-2	s2
parität	p<type>	n=keine e=gerade o=ungerade m=mark (immer 1) s=space (immer 0)	pn
duplex	d<type>	h=halb f=full	df
handshake	h<type>	3=3-Draht x=x-Draht	h3

termchar$

ist eine Funktion, die angibt, welches Zeichen zur Beendigung der Eingabe im letzten Eingabefeld verwendet wurde, auf das der Benutzer reagiert hat. Zum Beispiel liefert **termchar$** den Wert CHR$(13), wenn die Eingabe mit **<Return>** abgeschlossen wurde. Siehe auch: INPUT, **termpos**, **termchars$** und **inputpos**.
Ein Programmbeispiel findet sich auf der C-128 Demodiskette unter: "demo.termchar.80".

termchars(zeichen$)

ist eine Prozedur, die festlegt, welche Zeichen die Eingabe im nächsten Eingabefeld beenden können. Der Standardwert für **zeichen$** ist **CHR$(13)**, was bedeutet, dass die Eingabe durch Drücken der **<Return>**-Taste abgeschlossen wird.

Folgende Zeichen haben eine besondere Bedeutung:

Zeichen	Bedeutung
CHR$(128)	Beendet die Eingabe bei <Cursor links>, wenn sich der Cursor ganz links im Eingabefeld befindet.
CHR$(130)	Beendet die Eingabe bei <Cursor rechts>, wenn sich der Cursor ganz rechts im Eingabefeld befindet.
CHR$(132)	Beendet die Eingabe, wenn in der letzten Position des Eingabefeldes ein Zeichen eingegeben wird.

Die definierten Termchar-Zeichen bleiben aktiv, bis im Programm eine neue **termchars**-Prozedur ausgeführt wird oder das Programm erneut mit RUN gestartet wird.

Beispiel:
```
0010 USE system
0020 termchars(""13""27"")
0030 INPUT "Testing termchars ... ": svar$
0040 PRINT "Du hast <Return> oder <ESC> gedrückt!"
0050 END
```

termpos

ist eine Funktion, die angibt, an welcher Position sich der Cursor im Eingabefeld bei Beendigung der Eingabe befand. Der Wert 1 entspricht der ersten Position im Eingabefeld.

Siehe auch: **inputpos**, **termchars**, **termchar$** und INPUT.

textcolors(rahmen#,hintergrund#,text#)

ist eine Prozedur, mit der sich die Textfarben des aktuellen Bildschirms ändern lassen. Die Parameter rahmen#, hintergrund# und text# sind Ganzzahlen von 0 bis 15. Sie bestimmen die Farben für den Bildschirmrahmen, den Hintergrund und den Text, der auf dem Bildschirm angezeigt wird.

Hinweis: Auf dem 80-Zeichen-Bildschirm entspricht die Rahmendfarbe immer der Hintergrundfarbe.

Die Farbcodes sind im Anhangs zu finden.

textmode(modus#)

ist eine Prozedur, die den Wechsel zwischen 40- und 80-Zeichen-Bildschirm erlaubt.

- modus# = 0 bedeutet 40-Zeichen-Bildschirm
- modus# = 1 bedeutet 80-Zeichen-Bildschirm

Hinweis: Beim Umschalten auf den 80-Zeichen-Bildschirm wird der Inhalt des 40-Zeichen-Bildschirms gelöscht, es sei denn, **cpuspeed** ist auf 1 gesetzt oder das Grafikpaket ist aktiv.

textwindow(zeilenmin,zeilenmax,spaltenmin,spaltenmax)

ist eine Prozedur, die ein Textfenster festlegt und den Cursor in die linke obere Ecke dieses Fensters setzt. Man kann das Textfenster aufheben und wieder Zugriff auf den gesamten Textbildschirm erhalten, indem man <Alt-F> oder <Home><Home> drückt.

vdcpeek(register)

ist eine Funktion, mit der man direkt aus den Registern des VDC-chips für den 80-Zeichen-Bildschirm lesen kann.

vdcpoke(register,wert)

ist eine Prozedur, die es ermöglicht, in die Register des VDC-chips für den 80-Zeichen-Bildschirm zu schreiben.

DAS NEUE RAMFILES-PAKET

Das Paket ramfiles ist exklusiv für das C-128-COMAL-Modul verfügbar. Diese Innovation wurde möglich, da im Commodore 128 etwa 40 KB Speicherplatz ungenutzt waren. UniComal nutzte diesen Bereich für eine effiziente Form der Datenspeicherung.
Das Paket eröffnet vielfältige Möglichkeiten zur schnellen Datenverarbeitung. Nutzer können selbst entscheiden, wie Daten strukturiert werden. Innerhalb der 40 KB können bis zu 32 RAM-Dateien im selben Programm definiert werden:

RAM-Datei 1	RAM-Datei 2	…	RAM-Datei 32
Datensatz 1	Datensatz 1		Datensatz 1
Datensatz 2	Datensatz 2		Datensatz 2
…	…		…
…	…		…
Datensatz N_1	Datensatz N_2	…	Datensatz N_{32}

Jeder Datensatz kann mehrere Felder enthalten, die aus reellen Zahlen, Ganzzahlen, Bytes, Booleschen Variablen (einzelne Bits) oder Zeichenfolgen bestehen können. Beispiel: Sie möchten eine Datenbank mit folgenden Informationen erstellen:

Kundennummer	(Ganzzahl)
Kundenname	(String, max. 38 Zeichen)
Adresse	(String, max. 78 Zeichen)
Gesamtumsatz	(reelle Zahl)
Rabatt gewährt	(Ja/Nein – 1 Bit)
30-Tage-Kredit	(Ja/Nein – 1 Bit)

Die Struktur eines Datensatzes ist wie folgt:

Datensatz:	Ganzzahl	String 1	String 2	Reelle Zahl	Bit 1	Bit 2

Speicherbedarf pro Datensatz

Feld	**Größe**
Ganzzahl	2 Bytes
String 1 (Name)	40 Bytes (38 + 2 Längenbytes)
String 2 (Adresse)	80 Bytes (78 + 2 Längenbytes)
Reelle Zahl	5 Bytes
2 boolesche Werte	1 Byte (2 Bits)
Gesamt	**128 Bytes pro Datensatz**

Berechnung des Gesamtspeichers
Verwendete Bytes = (Anzahl der Datensätze × Datensatzlänge) + (Anzahl der Felder × 4) + 16
Beispiel für 300 Datensätze:
300 × 128 + 6 × 4 + 16 = 38.440 Bytes
Der verfügbare Speicher für RAM-Dateien beträgt insgesamt 40.896 Bytes.

Wie aus der folgenden alphabetischen Beschreibung der einzelnen Prozeduren und Funktionen des ramfiles-Pakets hervorgeht, kann die Struktur des Datensatzes mit der Prozedur createrf(1,321,"I S38 S78 R LL") definiert werden.
Im Folgenden wird jede ramfile-Prozedur und -Funktion beschrieben, oft ergänzt durch kurze Beispiele. Ein ausführlicheres Beispiel zur Verwendung von RAM-Dateien finden Sie auf der C-128-Demodiskette ("RAMFILES").

createrf(dateinummer#,anzahl_datensaetze#,beschreibung$)

Diese Prozedur dient zum Erstellen einer RAM-Datei mit der Nummer dateinummerr# (1–32). Der Dateizeiger wird auf das erste Feld des ersten Datensatzes gesetzt Diese Dateistruktur ähnelt in vielerlei Hinsicht einer direkten Datei auf einer Diskette, da Sie vorher festlegen müssen, wie viele Datensätze die Datei enthalten soll, (anzahl_datensaetze - maximal 32767) und wie die einzelnen Felder im Datensatz aussehen sollen (beschreibung$). Die Beschreibung ist eine Zeichenfolge, die Informationen zu den Feldern in jedem Datensatz enthält. Jedes Feld wird durch einen Buchstaben (Groß- oder Kleinschreibung) beschrieben, der die Felder wie folgt kennzeichnet:

"R" oder "r"	steht für eine reelle Zahl. Eine reelle Zahl belegt 5 Bytes im RAM.
"I" oder "i"	steht für eine Ganzzahl. Jede Zahl belegt 2 Byte.
"B" oder "b"	steht für ein Byte, das einen ganzzahligen Wert zwischen 0 und 255 annehmen kann.
"L" oder "I"	steht für eine logische (oder boolesche) Variable, d. h. ein einzelnes Bit (0 oder 1). Acht boolesche Variablen belegen 1 Byte.
"S" oder "s"	gefolgt von einer Zahl gibt die maximale Länge einer Zeichenfolge an (max. 32767). Beispielsweise bedeutet "S20", dass das Feld Platz für 20 Zeichen bietet. Beachten Sie bei der Berechnung des Speicherbedarfs, dass jedem Feld zwei Byte zur Angabe der Zeichenfolgenlänge zugewiesen sind.

Beispiele: **createrf(1,100,"s5s10s40")** erstellt die RAM-Datei Nummer 1 mit Platz für 100 Datensätze. Jeder Datensatz besteht aus drei Zeichenfolgen mit jeweils 5, 10 und 40 Zeichen.

createrf(2,50,"I S10 BBB R LLL") erstellt eine RAM-Datei (Dateinummer 2) mit Platz für 50 Datensätze. Jeder Datensatz besteht aus einer Ganzzahl, einer Zeichenfolge mit maximal 10 Zeichen, drei Bytes, einer reellen Zahl und drei logischen Variablen (Bits).

Ein kurzes Programmbeispiel finden Sie unter "eofrf" und ein ausführlicheres Beispiel zur Verwendung von RAM-Dateien finden Sie auf der Demo-Disk

deleteallrf

ist eine Prozedur, die alle RAM-Dateien im Arbeitsspeicher löscht. Die Daten einer RAM-Datei bleiben auch nach Beendigung eines Programms im RAM erhalten. Sie werden sofort gelöscht, wenn der Computer ausgeschaltet oder **"deleteallrf"** oder **"deleterf"** ausgeführt wird.

deleterf(Datei#)

ist eine Prozedur, die Daten in der RAM-Datei mit der Nummer Datei# löscht. Beachten Sie, dass Daten in einer RAM-Datei erst gelöscht werden, wenn der Computer ausgeschaltet wird. Wenn ein Programm, in dem eine RAM-Datei erstellt wird, mehr als einmal ausgeführt wird, z. B. während der Programmentwicklung, muss die Datei gelöscht werden, bevor eine Datei mit derselben Nummer erstellt werden kann. Siehe auch **"deleteallrf"**.

Beispiel: **deleterf(2)** löscht die RAM-Datei Nummer 2 aus dem Speicher.

Ein kurzes Programmbeispiel finden Sie unter "eofrf". Ein ausführlicheres Beispiel zur Verwendung von RAM-Dateien finden Sie auf der C-128-Demodiskette.

eofrf(Datei#)

ist eine Funktion. Beim Erstellen einer RAM-Datei wird ein Zeiger erstellt, der zunächst auf das erste Feld im ersten Datensatz zeigt. Dieser Zeiger bleibt auch beim Speichern oder erneuten Abrufen der Datei erhalten. Der Zeiger wird beim Schreiben oder Lesen eines einzelnen Felds auf das nächste Feld aktualisiert. Beim Lesen oder Schreiben des letzten Felds eines Datensatzes wechselt der Zeiger zum ersten Feld des nächsten Datensatzes. Beim Lesen oder Schreiben des letzten Felds im letzten Datensatz wird ein Dateiende-Flag gesetzt. Dieses Flag kann mit der Funktion "eofrf(Datei#)" analog zu den Funktionen EOF und EOD getestet werden. Ist das Flag gesetzt, führt weiteres Lesen oder Schreiben in die RAM-Datei zu einer Fehlermeldung "Ende der RAM-Datei".

Beispiel:

```
0010 USE ramfiles
0020 DIM text$(5) OF 10
0030 DATA "Volvo","Mercedes","Saab","Ford","Nissan"
0040
0050 nr:=0
0060 WHILE NOT EOD DO
0070     nr:+1
0080     READ text$(nr)
0090 ENDWHILE
0100
0110 deleterf(2)
0120 createrf(2,nr,"s10")
0130 nr:=1
0140 WHILE NOT eofrf(2) DO
0150     writestr(2,text$(nr))
0160     nr:+1
0170 ENDWHILE
0180
0190 posrf(2,1,1)
0200 REPEAT
0210     readstr(2,autos$)
0220     PRINT autos$
0230 UNTIL eofrf(2)
```

fieldtype$(Datei#,feld#)

ist eine String-Funktion, die den Beschreibungstext für die einzelnen Felder einer RAM-Datei zurückgibt.

Beispiel 1:

```
0010 USE ramfiles
0020 deleterf(1)
0030 createrf(1,50,"rbls10")
0040 print fieldtype$(1,1)
```

Das obige kleine Programm gibt "r" auf dem Bildschirm aus. Dies entspricht dem ersten Feldtyp in der Beschreibungszeichenfolge für RAM-Datei Nummer 1, der eine reelle Zahl ist. Die gesamte Beschreibungszeichenfolge kann mit dem folgenden Programm erstellt werden:

Beispiel 2:

```
0010 USE ramfiles
0020 beschreibung$:=""
0030 deleterf(1)
0040 createrf(1,50,"rbls10")
0050
0060 FOR feld:=1 TO inqrf(1,3) DO // anzahl Felder
0070 beschreibung$:=beschreibung$+fieldtype$(1,feld)
0080 ENDFOR feld
0090
0100 PRINT beschreibung$
```

freerf

ist eine Funktion, die die Anzahl der freien Bytes im von den RAM-Dateien belegten Speicherbereich zurückgibt. Beim Start sind 40896 Bytes frei.

Beispiel: Nach Ausführung des Programms in Beispiel 2 unter **"fieldtype$"** gibt **PRINT "freerf"** die Zahl 39914 zurück.

getnum(Datei#)

ist eine Funktion zum Abrufen einer Zahl aus der RAM-Datei Datei#. Die Funktion gibt die Zahl zurück, auf die der Dateizeiger zeigt, und aktualisiert den Dateizeiger auf das nächste Feld.

getrec$(Datei#,Datensatz#)

ist eine String-Funktion, die einen String im internen Format zurückgibt, der dem Eintrag Datensatz# in der RAM-Datei mit der Nummer Datei# entspricht. Der Dateizeiger wird nicht aktualisiert.

getstr$(Datei#)

ist eine Funktion, mit der ein String aus der Datei mit der Nummer Datei# abgerufen wird. Der Dateizeiger muss zunächst mit **posrf(Datei#,Datensatz#,feld#)** oder durch sequentielles Lesen oder Schreiben in die RAM-Datei korrekt positioniert werden. Die Fehlermeldung "Falscher RAM-Feldtyp" erscheint, wenn das Feld keinem String entspricht. Der Dateizeiger wird auf das nächste Feld aktualisiert.

inqrf(Datei#,nummer#)

ist eine Funktion, die die folgenden Werte zurückgibt, um das Format einer "unbekannten" RAM-Datei zu überprüfen:

nummer#:
- 1: Anzahl der reservierten Datensätze
- 2: Länge der einzelnen Datensätze
- 3: Anzahl der Felder in einem Datensatz
- 4: Aktuelle Datensatznummer
- 5: Aktuelle Feldnummer

Wenn die RAM-Datei Datei# nicht erstellt wurde, wird immer 0 zurückgegeben.

loadallrf(Dateiname$)

ist eine Prozedur, die alle RAM-Dateien von der Diskette abruft, die zuvor unter Dateiname$ gespeichert waren. Siehe auch "saveallrf".

Beispiel:
```
saveallrf("@0:alle Dateien")
loadallrf("alle Dateien")
```

loadrf(Datei#,Dateiname$)

ist eine Prozedur, mit der Sie die RAM-Datei mit der Nummer Datei#, die zuvor unter Dateiname$ gespeichert war, von der Diskette abrufen können. Alle vorherigen RAM-Dateien mit derselben Nummer müssen zuvor gelöscht werden. Ein Beispiel für eine Prozedur zum Abrufen einer RAM-Datei mit dem Namen "ramfile 1" finden Sie im Programm "RAMFILES" auf der C-128 COMAL Demodiskette.

posrf(Datei#,Datensatz#,Feldnummer#)

ist eine Prozedur, mit der der Benutzer den Dateizeiger beliebig in einem bestimmten Feld innerhalb eines bestimmten Datensatzes innerhalb einer bestimmten Datei positionieren kann, um RAM-Dateilese- oder -schreibvorgänge auszuführen. Siehe auch **"eofrf"**, **"getstr$"**, **"writenum"** und **"writestr"** für Beispiele zur Verwendung von **"posrf"**. Siehe auch das Demonstrationsprogramm "RAMFILES" auf der C-128 COMAL Demodiskette.

readnum(Datei#,wert)

ist eine Prozedur, mit der eine Nummer aus der Datei Datei# abrufen und in die Variable wert übertragen wird. Ein Programmbeispiel für **"readnum"** finden Sie unter **"posrf"**. Der Dateizeiger wird automatisch auf das nächste Feld aktualisiert.

readrec(Datei#,Datensatz#,String$)

ist eine Prozedur, mit der die Daten Datensatz# aus der RAM-Datei Datei# abrufen und in den REF-Parameter String$ übertragen können.
Beachten Sie, dass der Datensatz gegebenenfalls im internen Format vorliegt. Das bedeutet, dass Strings, Integer, Bits, Bytes und reelle Zahlen in der Form angezeigt werden, in der sie in einer RAM-Datei vorliegen. Der Dateizeiger wird nicht auf das nächste Feld aktualisiert.

readstr(Datei#,String$)

ist eine Prozedur, mit der ein String$ abgerufen werden kann, der zuvor mit **"writestr"** gespeichert wurde. Ein Programmbeispiel für die Verwendung von **"readstr"** finden Sie unter "**posrf"**. Der Dateizeiger wird automatisch auf das nächste Feld aktualisiert.

saveallrf(Dateiname$)

ist eine Prozedur, die alle vorhandenen RAM-Dateien auf der Festplatte unter dem Namen Dateiname speichert. Alle Dateien können mit loadallrf (Dateiname$) wieder abgerufen werden. Es ist auch möglich, einzelne RAM-Dateien zu speichern. Siehe **"saverf"** und **"loadrf"**. Beachten Sie, dass **"loadrf"** (Datei#, Dateiname$) nicht zum Abrufen einer Datei verwendet werden kann, wenn **"saveallrf"** zum Speichern verwendet wurde. Gleiches gilt für **"saverf"** und **"loadallrf"**. Das Dateiformat ist beim Speichern einer einzelnen oder aller Dateien unterschiedlich. Bei nicht übereinstimmendem Format wird die Fehlermeldung "Falscher Typ in LOAD-Datei" angezeigt.

saverf(Datei#,Dateiname$)

ist eine Prozedur, die die Datei mit der Nummer Datei# unter dem Dateinamen Dateiname$ auf der Festplatte speichert.

Beispiel: saverf(1,"adressen") speichert die RAM-Datei Nummer 1 auf der Diskette unter dem Dateinamen "adressen".

writenum(Datei#,wert)

ist eine Verfahren zum Speichern eines numerischen Werts in einer RAM-Datei. Der Dateizeiger muss zunächst mit **"posrf"** oder mittels sequentiellem Lesen/Schreiben positioniert werden, bevor in die Datei geschrieben werden kann. Nach der Operation wird der Dateizeiger auf das nächste Feld aktualisiert.

Beispiel:

```
0010 USE ramfiles
0020 deleterf(1); createrf(1,10,"ir")
0030
0040 FOR Zahl#:=1 TO 10 DO
0050     writenum(1 ,Zahl#)
0060     writenum(1 ,SQR(Zahl#))
0070 ENDFOR Zahl#
0080 posrf(1,1,1)
0090 REPEAT
0100     readnum(1,wert)
0110     readnum(1,wurzel)
0120     PRINT wert;wurzel
0130 UNTIL eofrf(1)
```

writerec(Datei#,Datensatz#,String$)

ist eine Prozedur, mit der ganze Datensätze gleichzeitig in eine RAM-Datei geschrieben werden können. Der übergebene String$ wird in ein internes Format gepackt, und LEN(String$) ist die Datensatzlänge. Die umgekehrte Operation wird mit der Prozedur **readrec**(Datei#,Datensatz#,String$) ausgeführt. Die mit **"readrec"** gelesene Textvariable muss mindestens diese Länge haben. Der Dateizeiger wird durch diese Prozedur nicht verändert.

writerefrec(Datei#,Datensatz#,String$)

ist eine Prozedur, mit der ein ganzer Datensatz gleichzeitig in eine RAM-Datei geschrieben werden kann. Die Verwendung dieser Prozedur anstelle von **"writerec"** spart Zeit und Speicherplatz. LEN(String$) ist die Datensatzlänge. Beachten Sie, dass die umgekehrte Operation mit **readrec**(Datei#,Datensatz#,String$) ausgeführt werden kann. Der gelesene String muss mindestens so lang sein wie die Datensatzlänge. Die Operation verändert den Dateizeiger nicht.

writerefstr(Datei#,Datensatz#,String$)

ist eine Prozedur zum Speichern eines String$ in einer RAM-Datei. Die Verwendung von **"writerefstr"** im Vergleich zu **"writestr"** spart Zeit und Speicherplatz. Die umgekehrte Operation kann mit der Prozedur **"readstr"** durchgeführt werden. Der Dateizeiger wird auf das nächste Feld aktualisiert.

writestr(Datei#,String$)

ist eine Prozedur zum Speichern eines Strings in einer RAM-Datei. Zuvor muss **"posrf"** ausgegeben werden, damit der Dateizeiger auf den richtigen Datensatz und das richtige Feld zeigt. Nach der Operation wird der Dateizeiger auf das nächste Feld aktualisiert.

Beispiel:

```
0010 USE ramfiles
0020 deleterf(1);createrf(1,6,"s8s10")
0030
0040 DATA "Boeing","Lockheed","Douglas","Rockwell"
0045 DATA "Cessna","Piper"
0050 DATA " 747"," Electra"," DC-3"," Commander"
0055 DATA " 172"," Cherokee"
0060
0070 FOR post#:=1 to 6 DO
0080     posrf(1,post#,1)
0090     READ fabrik$
0100     writestr(1,fabrik$)
0110 ENDFOR post#
0120
0130 FOR post#:=1 TO 6 DO
0140     posrf(1,post#,2)
0150     READ fly$
0160     writestr(1,fly$)
0170 ENDFOR post#
0180
0190 posrf(1,1,1)
0200 WHILE NOT eofrf(1) DO
0210     readstr(1,fabrik$);readstr(1,fly$)
0220     PRINT fabrik$,fly$
0230 ENDWHILE
```

Die folgenden Fehlermeldungen sind neu und beziehen sich auf das Ramfiles-Paket:

37:	ram file already created	ram-datei schon angelegt
38:	ram file not created	ram-datei nicht angelegt
39:	invalid ram file number	unerlaubte ram-datennummer
40:	invalid ram record number	unerlaubte ram-satznummer
41:	error in descriptor string	fehler in feldtyp-beschreibung
42:	wrong ram field type	falscher ram-feldtyp
43:	invalid ram field number	unerlaubte ram-feldnummer
44:	end of ram file	ende der ram-datei
45:	wrong file type in load file	falscher typ in load-datei

WEITERE SCHLÜSSELWÖRTER

Neben dem Systempaket und dem neuen RAM-Dateipaket wurden folgende Teile von C-64 COMAL verbessert, erweitert oder geändert:

* COMAL-Kernel
* Grafikpaket
* Sprite-Paket
* Schriftartenpaket

Ein Teil dieser Änderungen ist darauf zurückzuführen, dass die neue COMAL-Version 2.02 die 80-Zeichen-Anzeige des C-128 nutzen kann.
Darüber hinaus ergeben sich einige Änderungen daraus, dass die Zeichen , (Komma) und ; (Semikolon) nach der letzten internationalen Standardisierungssitzung in COMAL eine neue Bedeutung erhalten haben.
Auch die IN-Anweisung wurde bei diesem Treffen, das im September 1986 in Dänemark stattfand, leicht geändert.

Schlüsselwörter des COMAL-Kernal

BASIC

Mit diesem Befehl kann man COMAL verlassen und zum Commodore BASIC 7 wechseln. Es ist möglich, anschließend wieder nach COMAL zurückzukehren, indem man den Befehl SYS 14*4096 eingibt.
Falls die BANK in BASIC geändert wurde, muss BANK 15 vor dem SYS-Befehl eingegeben werden.

Beispiel:	BASIC	wechselt von COMAL zu BASIC.
	SYS 14*4096	wechselt wieder zu COMAL zurück.

get#(filenr#,anzahl_zeichen#)

ist eine Funktion aus dem COMAL-Kern. Sie wurde so geändert, dass beim Einsatz von GET$ auf eine Datei mit der Dateinummer filenr#, die zur seriellen Schnittstelle (“sp:”) geöffnet wurde, um anzahl_zeichen# zu lesen, nur so viele Zeichen gelesen werden, wie tatsächlich im RS-232-Empfangspuffer vorhanden sind.
Das bedeutet, dass Programme nicht mehr “hängen bleiben”, wenn sich kein Zeichen im Puffer befindet.

IN

ist ein Schlüsselwort, das in C-128 COMAL eine leicht geänderte Bedeutung erhalten hat.
Wird der Ausdruck **string1$ IN string2$** ausgewertet, wobei string1$ ein leerer String ("") ist, so wird das nun als FALSE (0) ausgewertet.
In C-64 COMAL wurde dieser Ausdruck dagegen als LEN(string2$)+1 ausgewertet, also als TRUE (1).

Beispiel:
```
INPUT "Sollen wir fortfahren (j,n)?":antwort$
IF antwort$ IN "jJ" THEN DELETE "program"
```

In C-64 COMAL würde das Programm DELETE "programm" ausführen, auch wenn der Benutzer nicht j oder J eingibt, sondern nur RETURN drückt, denn antwort$ wäre dann ein leerer String.
Dies geschieht in C-128 COMAL nicht mehr, da antwort$ IN "jJ" nun als FALSE ausgewertet wird, wenn antwort$="" ist. Jetzt muss der Benutzer ausdrücklich j oder J eingeben, damit der THEN-Teil ausgeführt wird.

INPUT

ist ein Schlüsselwort aus dem COMAL-Kern, das mit dem C-128-Modul erweitert wurde.
Attribute (Unterstreichung, invertierter Text und Blinken) bleiben in einem Eingabefeld erhalten. Beispielsweise bleibt invertierter Text auch dann erhalten, wenn das Feld mit **<Clr/Home>** gelöscht wird. Ähnlich wie bei der PRINT-Anweisung verhalten sich nun auch das Komma (,) und das Semikolon (;) anders, wenn sie am Ende einer INPUT-Anweisung verwendet werden.

Beispiel:
```
0010 PAGE
0020 INPUT "Ist das Wetter ok (j/n)? ":antwort$;
0030 IF antwort$=""j THEN
0040     PRINT "Okay, lass uns gehen."
0050 ELSE
0060     PRINT „Dann lass uns zu Hause bleiben.“
0070 ENDIF
```

Ein Semikolon (;) bewirkt jetzt immer, dass genau ein Leerzeichen nach der Eingabe des Benutzers ausgegeben wird, es sei denn, ZONE wurde auf einen anderen Wert gesetzt.
Ein Zeilenumbruch (Carriage Return) wird dabei nicht ausgeführt, und der Text nach der Benutzereingabe erscheint in derselben Zeile.

In das Systempaket wurden zwei Prozeduren eingeführt, um die Arbeit mit Eingaben zu erleichtern:

- inputpos setzt den Cursor an eine bestimmte Startposition innerhalb eines Eingabefeldes.
- termchars legt fest, welche Zeichen eine Eingabe in einem Eingabefeld beenden können.

Außerdem gibt es zwei neue Funktionen in der Systembibliothek:

- termpos gibt an, an welcher Position im Eingabefeld sich der Cursor befand, als die Eingabe abgeschlossen wurde.
- termchar$ gibt den Zeichencode des Eingabe-Abschlusszeichens in einem Eingabefeld zurück.

Siehe zu diesen Funktionen auch die jeweiligen Abschnitte sowie das Programm "demo.termchar.80" auf der C-128 Demodiskette.

PRINT

ist ein Schlüsselwort, das als Anweisung oder Befehl verwendet werden kann, um Daten auf einem Ausgabegerät wie einem Bildschirm oder Drucker zu drucken. In Verbindung mit der Verwendung der PRINT-Anweisung haben Komma (,) und Semikolon (;) eine etwas andere Bedeutung erhalten. Siehe die Abschnitte über INPUT und ZONE für eine genauere Erläuterung sowie Programmbspiels.

SIZE

ist ein Befehl, der verwendet wird, um zu zeigen, wie der Speicher für Programme, Daten und RAM-Dateien genutzt wird:

Beispiel: SIZE

```
prog      data      free
00065     0000      40889
ramfiles:
used      free
00982     39914
```

ZONE [Wert]

ist ein Schlüsselwort, das als **Anweisung**, **Befehl** oder **Funktion** verwendet werden kann. ZONE hat in C-128 COMAL eine leicht geänderte Bedeutung im Vergleich zu C-64 COMAL. Jetzt ist das Semikolon (;) das Trennzeichen (anstatt des Kommas), auf dem ZONE basiert. Der Standardwert ist jetzt 1 (früher 0), sodass das Semikolon dieselbe Bedeutung wie zuvor hat, wenn ZONE nicht gesetzt wurde. Wenn ZONE auf einen Wert gesetzt wird, gibt dieser die Anzahl der Plätze für jede Tabulatorstellung an.

Das Komma (,) wirkt jetzt immer nur als Trennzeichen und wird nicht mehr durch ZONE beeinflusst.

Beispiel:
```
0010 meine_zone:= ZONE // ZONE kann als Funktion verwendet werden
0020 ZONE 10         // oder als Anweisung.
0030
0040 FOR nr:= 1 TO 5 DO
0050     PRINT nr;nr;nr
0060 ENDFOR nr
0070
0080 ZONE meine_zone
```

Diese Änderung wurde vorgenommen, um ein Trennzeichen — nämlich das Komma (,) — zu garantieren, das keine Bedeutung hat. Dies ist besonders wichtig, da das Komma auch als Datentrennzeichen bei ASCII-Dateioperationen verwendet wird. Früher konnte es passieren, dass unerwartete Leerzeichen in einer Datei auftauchten, wenn eine Anweisung wie PRINT FILE a$, b$, c$ ausgeführt wurde, während ZONE nicht null war. In C-128 COMAL kann dies nun nicht mehr passieren.
Die Änderung wurde von der COMAL-Standardisierungsgruppe im September 1986 beschlossen.

SCHLÜSSELWÖRTER AUS DEM GRAFIKPAKET

textcolor (farbe#)
textborder (farbe#)
textbackground (farbe#)

Diese Prozeduren aus dem Grafikpaket, die jeweils eine farbe# (0-15) als Parameter verwenden, funktionieren nur auf dem 40-Zeichen-Bildschirm. Beachte, dass alle anderen Optionen von der Tastatur und den Systempaketen auch für den aktuellen Bildschirm gelten.
Die Prozedur **textcolors** aus dem Systempaket sollte verwendet werden, wenn man die Textfarbe auf dem 80-Zeichen-Bildschirm ändern möchte. Siehe die letzte Seite dieses Anhangs für eine Übersicht der Farbcodes.

SCHLÜSSELWÖRTER AUS DEM SPRITEPAKET

getshape$(spritenr#)

ist eine String-Funktion im Sprite-Paket, die einen String mit dem Inhalt des Sprites mit der Nummer spritenr# zurückgibt. Sie ist die Umkehroperation von define(spritenr#, sprite$).

SCHLÜSSELWÖRTER AUS DEM FONTPAKET

discardfont

ist eine Prozedur im Font-Paket. Sie stellt den Standard-Zeichensatz wieder her, auch wenn zuvor **keepfont** ausgeführt wurde. Beachte, dass im 40-Zeichen-Modus der Arbeitsspeicher, der für den Font verwendet wurde, erst nach NEW oder DISCARD freigegeben wird.

selectfont(zeichensatz#)

ist eine Prozedur im Font-Paket, die den Wechsel zu Font Nummer zeichensatz# auf dem aktuellen Bildschirm bewirkt. Zulässige Werte für zeichensatz# sind 0 und 1 für den 80-Zeichen-Bildschirm sowie 0, 1, 2 und 3 für den 40-Zeichen-Bildschirm (jedoch nicht 0 oder 1, wenn kein benutzerdefiniertes Zeichensatz vorhanden ist).

Beispiel:
```
USE font
USE system
textmode(1)          // wählt den 80-Zeichen-Bildschirm
loadfont("font1")    // lädt eine Font-Datei von der Diskette
selectfont(1)        // aktiviert Font 1 zur Verwendung
```

Anhang A: PROBLEMLÖSUNG

DAS PROGRAMM LÄUFT, ABER ES IST NICHTS AUF DEM BILDSCHIRM ZU SEHEN:

Überprüfen Sie, ob alle erforderlichen Kabel angeschlossen sind. Es kann sein, dass das Programm für den Betrieb auf einem 80-Zeichen-Bildschirm ausgelegt ist, der Computer aber an einen 40-Zeichen-Bildschirm angeschlossen ist – oder umgekehrt. Falls möglich, kann man auf den anderen Bildschirm umschalten. Beachten Sie auch, dass der 40-Zeichen-Bildschirm nicht verwendet werden kann, wenn man mithilfe der Systempaket-Prozedur cpuspeed(2) eine Taktfrequenz von 2 MHz gewählt hat. Verwenden Sie stattdessen cpuspeed(0) (was beim Starten der Standardwert ist).

Wenn man nur einen der beiden Bildschirmtypen besitzt, können folgende Prozeduren verwendet werden, um den Bildschirmtyp auszuwählen:

USE system
textmode(1) wählt den 80-Zeichen-Bildschirm
textmode(0) wählt den 40-Zeichen-Bildschirm

Man kann eine dieser Anweisungen eventuell in eine Startprozedur im Programm einbauen, um sicherzustellen, welcher Bildschirm verwendet wird.

EINGABE ODER AUSGABE WIRD NICHT RICHTIG AUF DEM BILDSCHIRM ANGEZEIGT:

Es ist möglich, dass das Programm für einen 80-Zeichen-Bildschirm geschrieben wurde, aber auf einem 40-Zeichen-Bildschirm ausgeführt wird – oder umgekehrt. Bearbeiten Sie das Programm oder wechseln Sie den Bildschirm. Verwenden Sie ggf. textmode(1) oder textmode(0), um sicherzustellen, dass das Programm immer auf dem richtigen Bildschirm läuft.

DIE AKTIVITÄTSLEUCHTE DES DISKETTENLAUFWERKS BLINKT:

Wenn die Aktivitätsleuchte des Diskettenlaufwerks blinkt und das Laufwerk arbeitet, ohne dass sofort etwas geschieht, kann das daran liegen, dass das System auf einen anderen Laufwerksmodus umschaltet. Das alte 1541-Diskettenlaufwerk kann problemlos mit dem C-128 und dem neuen COMAL-Modul verwendet werden. Andererseits profitiert man sehr davon, Zugriff auf ein 1570 (ein schnelles, einseitiges Diskettenlaufwerk) oder ein 1571 (schnelles, doppelseitiges Laufwerk) zu haben.

Wenn man C-64-Disketten mit COMAL-Programmen besitzt, die man gern auf C-128 COMAL übernehmen möchte, kann man die alten Disketten auch im schnellen Laufwerk verwenden. Das Diskettenlaufwerk muss jedoch kurz arbeiten (die Aktivitätsleuchte blinkt), um zu erkennen, dass die Diskette auf einem 1541 formatiert wurde, bevor ein CATALOG angezeigt oder Programmdateien geladen werden können.

Es ist in C-128 COMAL möglich, zwischen dem 1570/1571-Modus und dem 1541-Modus zu wechseln – und umgekehrt. Zu diesem Zweck kann man folgende Befehle verwenden:

PASS "u0>m0" wechselt vom 1570/1571-Modus in den 1541-Modus
PASS "u0>m1" wechselt vom 1541-Modus in den 1570/1571-Modus

PROGRAMMZEILEN WERDEN NICHT ZUSAMMENHÄNGEND AUF DEM BILDSCHIRM ANGEZEIGT:

Wenn man ein Programm bearbeiten möchte, ist es in COMAL Version 2.02 möglich, den Cursor auf eine Programmzeile zu setzen und Änderungen vorzunehmen. Wenn man dann **<Return>** drückt, wird die Zeile in das Programm übernommen.Man kann auch **<Ctrl-A>** drücken, um eine „Zeile zusammenzuziehen“, falls sie durch zusätzlich eingefügte Leerzeichen umgebrochen wurde, etwa weil sie beim Auflisten mehr als eine Zeile auf dem Bildschirm eingenommen hat.

Wenn ein Programm ein Scrollen des Bildschirms verursacht, können Situationen entstehen, in denen ein Teil einer umgebrochenen Programmzeile nicht mehr als zusammenhängend mit dem Rest der Zeile erkannt wird. Dieses Problem lässt sich beheben, indem man den Cursor auf den Anfang der Zeile (dort, wo die Zeilennummer steht) bewegt und **<Ctrl-A>** drückt.

EINIGE CTRL-, ALT- ODER ESC-CODES FUNKTIONIEREN NICHT:

UniComal nutzt einige der eingebauten Funktionen des C-128. Wenn man eine der ersten Versionen des Computers besitzt, waren manche dieser Tastatursequenzen noch nicht implementiert. Dies betrifft z. B. **<Alt-Inst>**, hier kann man stattdessen **<Esc><A** verwenden.

Anhang B – Alt-, Ctrl- und ESC-Codes

Ihre Commodore-128-Tastatur bietet zusätzliche Möglichkeiten zur Steuerung des Bearbeitungsvorgangs und der Nutzung der Textbildschirme. Vielleicht haben Sie bereits die Tastenkombinationen **<Alt CRSR hoch>** (scrollen des Listings nach unten) und **<Alt CRSR runter>** (scrollen des Listings nach oben) ausprobiert – eine nützliche Hilfe, wenn man mit einem Programmausdruck auf dem Bildschirm arbeitet.

Einige der folgenden Kombinationen können direkt von der Tastatur ausgeführt werden (**d**), einige aus einem Programm heraus (**p**) und einige auf beide Arten.

Von der Tastatur:

Um z. B. **<Ctrl-R>** von der Tastatur auszuführen, halten Sie die **<Ctrl>**-Taste gedrückt und drücken gleichzeitig **<R>**. Versuchen Sie z. B. Folgendes:

Beispiel: <Ctrl-R>print<Return> Der Text „print“ wird in umgekehrter Schrift (Inverse) dargestellt.

Dasselbe gilt für die Alt-Codes wie z.B. **<Alt-I>** : Halten Sie die **<Alt>**-Taste gedrückt und drücken Sie die **<I>**-Taste, um eine leere Zeile auf dem Bildschirm einzufügen.

Aus einem Programm:

Wenn man Inversschrift in einem Programm verwenden möchte, muss man nur dafür sorgen, dass , **<Ctrl-R>** entsprechend **CHR$(18)**, direkt vor dem Text steht, der in Inversschrift erscheinen soll. Die Inversschrift wird mit **CHR$(146)** beendet. Ein Zeilenumbruch beendet die Inversschrift ebenfalls.

Beispiel:
```
0010 inv$=CHR$(18); nor$=CHR$(146)
0020 PRINT inv$+"BEISPIELTEXT"+nor$+" 1 2 3 4"
```

ÄNDERUNGEN GEGENÜBER C-64 COMAL

Neuer Code:	Alter Code:	Funktion:
<Ctrl-H>	<Ctrl-B>	bewegt den Cursor ein Wort zurück.
<Esc><Q>	<Ctrl-K>	löscht den Rest einer Zeile.
<Esc><K>	<Ctrl-L>	bewegt den Cursor an das Zeilenende.
<Ctrl-G>	<Ctrl-X>	ändert die Rahmenfarbe.

Alt-CODES

Beachten Sie, dass die Alt-Codes nur direkt über die Tastatur eingegeben werden können.

<Alt-CRSR-links>	**d**	Bewegt den Cursor ein Wort zurück.
<Alt-CRSR-rechts>	**d**	Bewegt den Cursor ein Wort vorwärts.
<Alt-CRSR-hoch>	**d**	Scrollt eine Liste nach unten.
<Alt-CRSR-runter>	**d**	Scrollt eine Liste nach oben.
<Alt-Inst	**d**	Wechselt zwischen Einfüge- und Ersetzungsmodus. Der Einfügemodus wird durch eine nicht blinkende Markierung angezeigt. (Nicht bei frühen C-128-Modellen.)
<Alt-Del>	**d**	Entfernt das Zeichen unter dem Cursor.
<Alt-T>	**d**	Definiert die obere linke Ecke des Fensters.
<Alt-B>	**d**	Definiert die untere rechte Ecke des Fensters.
<Alt-F>	**d**	Schaltet auf Vollbild als Fenster um.
<Alt-I>	**d**	Fügt eine Zeile p in den Bildschirm ein.
<Alt-D>	**d**	Löscht eine Zeile auf dem Bildschirm.
<Alt-E>	**d**	Löscht den Rest des Bildschirms.
<Alt-R>	**d**	Löscht eine Zeile aus dem Programm und vom Bildschirm.
<Alt-N>	**d**	Fügt eine neue leere Programmzeile mit der Zeilennummer +l ein.

Ctrl-CODES

<Ctrl-A>	1	**d**	listet eine Programmzeile auf. Gut geeignet zum Rückgängigmachen einer Zeile, wenn eine Korrektur rückgängig gemacht wurde.
<Ctrl-B>	2	**d,p**	startet die Unterstreichung (nur) auf dem 80-Zeichen-Bildschirm.
<Ctrl-C>	3	**d**	entspricht **<Run/Stop>**.
<Ctrl-D>	4	**d**	gibt die Grafikseite aus (siehe Hinweis 1).
<Ctrl-E>	5	**d,p**	ändert die Cursorfarbe in Weiß.
<Ctrl-F>	6	**d**	bewegt den Cursor ein Wort vorwärts.
<Ctrl-G>	7	**d**	ändert die Rahmenfarbe auf dem 40-Zeichen-Bildschirm.
<Ctrl-G>	7	**p**	Glocke.
<Ctrl-H>	8	**d**	Bewegt den Cursor ein Wort zurück.
<Ctrl-I>	9	**d,p**	Tabulator.
<Ctrl-J>	10	**d,p**	Bewegt den Cursor in eine neue Zeile (Zeilenvorschub).
<Ctrl-K>	11	**d,p**	Sperrt <C=><Shift> auf einem 40-Zeichen-Bildschirm.
<Ctrl-L>	12	**d,p**	Entsperrt <C=><Shift> auf einem 40-Zeichen-Bildschirm.
<Ctrl-M>	13	**d,p**	Erzeugt einen Wagenrücklauf und eine neue Zeile.
<Ctrl-N>	14	**d,p**	Setzt Klein-/Großbuchstaben (auf einem 40-Zeichen-Bildschirm).

<Ctrl-O>	15	**d,p**	Setzt blinkende Zeichen (nur auf einem 80-Zeichen-Bildschirm).
<Ctrl-P>	16	**d**	Sendet eine Kopie des aktuellen Textbildschirms an den Drucker (siehe Hinweis 2).
<Ctrl-Q>	17	**d,p**	Bewegt den Cursor eine Zeile nach unten.
<Ctrl-R>	18	**d,p**	Aktiviert invertierten Text auf dem aktuellen Bildschirm.
<Ctrl-S>	19	**p**	Bewegt den Cursor zu Zeile 1, Zelle 1. <Home>.
<Ctrl-T>	20	**d,p**	Löscht ein Zeichen <Del>.
<Ctrl-U>	21	**d**	Deaktiviert grafische Funktionstasten.
<Ctrl-V>	22	**d**	Führt Textfarben(6,6,1) aus – blauer Bildschirm mit weißem Text auf dem aktuellen Bildschirm.
<Ctrl-W>	23	**d**	Führt Textfarben(1 l ,l5,0) auf dem aktuellen Bildschirm aus.
<Ctrl-X>	24	**d,p**	Tabstopps ein-/ausschalten.
<Ctrl-Y>	25	**d**	Ändert die Hintergrundfarbe.
<Ctrl-Z>	26	**d,p**	Ändert die Standardfarben.
<Ctrl-[>	27	**d,p**	<ESC>.
<Ctrl-]>	28	**d,p**	Bewegt den Cursor nach rechts.

Hinweis 1: <Ctrl-D> gilt nur für Commodore MPS-801-kompatible Drucker, bei denen ein Grafik-Dump möglich ist.

Hinweis 2: Wenn <Ctrl-P> den aktiven Textbildschirm ohne korrekten Zeilenumbruch ausgibt, versuchen Sie es mit **USE system** und der Prozedur **setprinter("lp:/l+").**

Esc-CODES

Alle C-128-Escape-Codes können während der Bearbeitung verwendet werden, mit Ausnahme von <Esc><L> und <Esc><M>, die das Scrollen des Bildschirms ein- und ausschalten, sowie <Esc><E> und <Esc><F>, die das Blinken der Markierung ein- und ausschalten. Beachten Sie, dass die ESC-Codes – im Gegensatz zu den Ctrl-Codes – als Tastenfolge eingegeben werden. Um beispielsweise vom 40-Zeichen- zum 80-Zeichen-Bildschirm oder zurück zu wechseln, geben Sie <ESC> und dann <X> ein.

<Esc><O>	Beendet Einrückung und Einfügemodus.
<Esc><Q>	Löscht bis zum Ende der aktuellen Zeile.
<Esc><P>	Löscht bis zum Anfang der aktuellen Zeile.
<Esc><@>	Löscht bis zum Ende des Bildschirms.
<Esc><J>	Bewegt den Cursor an den Anfang der aktuellen Zeile.
<Esc><K>	Bewegt den Cursor an das Ende der aktuellen Zeile.
<Esc><A>	Startet die Auto-Einfügefunktion.
<Esc><C>	Beendet die Auto-Einfügefunktion.
<Esc><D>	Löscht die aktuelle Zeile.
<Esc><I>	Fügt eine Zeile ein.
<Esc><Y>	Setzt den Standard-Tabstopp (8 Leerzeichen).
<Esc><Z>	Löscht alle Tabstopps.
<Esc><V>	Scrollt den Bildschirm nach oben.
<Esc><W>	Scrollt den Bildschirm nach unten.
<Esc><G>	Aktiviert die Glocke (siehe <Strg-G>).
<Esc><H>	Deaktiviert die Glocke, sodass sie nicht auf <Strg-G> reagiert.
<Esc><E>	Unterdrückt das Blinken des Cursors.
<Esc><F>	Löscht den Cursor zum Blinken.
<Esc><B>	Setzt den unteren Rand des Bildschirmfensters auf die Cursorposition.
<Esc><T>	Setzt den oberen Rand des Bildschirmfensters auf die Cursorposition.
<Esc><X>	Schaltet zwischen 40- und 80-Zeichen-Anzeige um.

Folgendes gilt nur für den 80-Zehntel-Bildschirm:

<Esc><U>	schaltet den Cursor unter.
<Esc><S>	schaltet den Cursor in den Blockmodus.
<Esc><R>	setzt den Bildschirm auf Negativdruck.
<Esc><N>	setzt den Bildschirm zurück auf Normaldruck.

Alle ESC-Codes können während der Programmausführung verwendet werden.

Anhang C: MASCHINENCODE-HANDBUCH

Dies ist eine Ergänzung zu Kapitel 8 von COMAL für den Commodore 64, das sich mit der Verknüpfung eines Maschinencodeprogramms mit COMAL Version 2.02 für den Commodore 128 befasst. Diese Ergänzung beschreibt nur die Unterschiede zwischen dem C-64 und dem C-128.

AUFTEILUNG DES RAM:

RAM-Bank 0:

0–8 KB in diesem Bereich befinden sich die Systemvariablen des Kernels, COMAL, der Prozessor-Stack und der Bildschirmspeicher (im Bereich 1–2 KB liegt der 40-Zeichen-Bildschirm-Speicher).
8–48 KB dieser bereich enthält den Speicher für COMAL-Programme, die Variablentabelle sowie den Stack. Hier können auch Pakete untergebracht werden, was jedoch auf Kosten des Benutzerspeichers geht. Falls Zeichensätze verwendet werden, werden diese für den 40-Zeichen-Bildschirm im Bereich 43–48 KB plaziert.
48–64 KB dieser Bereich kann für Maschinensprachenpakete verwendet werden, die keinen Benutzerspeicher belegen (der Bereich von $FFOO bis $FFFF darf jedoch nicht verwendet werden).

RAM-Bank 1:

0–8 KB dies ist der gemeinsamerRAM Bereich(Common RAM Area), d. h. dieser Bereich ist auch in RAM 0 adressierbar.
8–48 KB dieser Bereich wird für die RAM-Dateien verwendet.
48–64 KB dieser bereich wird für Grafik sowie diverse Tabellen (z. B. für Bildzuordnungen usw.) genutzt.

Der Ein-/Ausgabebereich (I/O) befindet sich wie beim C-64 im Bereich 52–56 KB.

FOLGENDE ROMs SIND IM COMMODORE 128 ENTHALTEN:

16–48 KB: BASIC-Interpreter
48–52 KB: Editor (für 40-/80-Zeichen-Bildschirme)
52–56 KB: Standard-Doppelzeichensatz
56–64 KB: Kernel

Das COMAL-Modul ist in 6 Seiten zu je 16 KB unterteilt, die sich alle im Adressbereich 48–64 KB befinden. Durch Bankswitching wird erreicht, dass ein 96-KB-System nur 16 KB Speicherplatz im Commodore 128 belegt.
Hinweis: Das COMAL-Modull liegt im gleichen Adressbereich wie der EDITOR und KERNEL.

SPEICHERVERWALTUNG:

Der Speicherbereich des C-128 wird von einer speziellen MMU (Memory Management Unit) gesteuert. Die Steuerung der Modul-Bänke erfolgt analog zum C-64, allerdings wurde der Port (OVERLAY) geändert und ist nun nicht mehr nur beschreibbar Der Inhalt kann nun auch gelesen werden.

MODULSTRUKTUR:

Die Module sind exakt wie in der C-64-Capsule aufgebaut, jedoch wurde .lib c64symb in .lib c128symb geändert (.lib c128symb entspricht .lib c128symbl + .lib c128symb2 ohne Kommentare usw.). Außerdem lautet <Startadresse> nun typischerweise $C000 statt $8009. .byte <map> gibt nun Speicherbereiche an, und die Bedeutung der verschiedenen Map-Werte ist in der folgenden Tabelle aufgeführt. Die gleichen Speicherbereiche gelten auch für SETPAGE im Systempaket:

Wert	Speicherbereich(memory map)
1	RAM 0
2	RAM 0 + I/O
3	RAM 1
4	RAM 1 + I/O
5	COMAL-Modul + RAM 0
6	COMAL-Modul + RAM 0 + I/O
7	COMAL-Modul + RAM 1
8	COMAL-Modul + RAM l + I/O
9	COMAL-Modul + RAM 2
1O	COMAL-Modul + RAM 2 + I/O
11	COMAL-Modul + RAM 3
12	COMAL-Modul + RAM 3 + I/O
13	KERNEL + RAM 0 + I/O
14	KERNEL + RAM 0 + CHARROM
15	KERNEL + RAM 0 + I/O + BASIC + MONITOR

Von diesen Speicherbereichen können alle als Parameter für SETPAGE verwendet werden – aber nur die Bereiche, die RAM 0 betreffen, dürfen als <MAP> in Modulen verwendet werden. Wenn DEFPAG gleich 2 ist (also RAM 0 + I/O), muss man beachten, dass der KERNEL in diesem Bereich nicht aktiv ist. Die wichtigsten Routinen können jedoch über indirekte Aufrufe verwendet werden, die in C128SYMB definiert sind. Wenn der Standardbereich (2) verwendet wird, darf Maschinencode nicht im Bereich $D000–$DFFF (I/O-Bereich) abgelegt werden. Wenn kein Zugriff auf I/O-Ports erforderlich ist, kann MAP 1 verwendet werden – ohne Einschränkungen in diesem Bereich.

Bevor COMAL-Routinen wie LOAD und STORE aufgerufen werden, muss die Routine SPAGEX aufgerufen werden. Der Akku muss dabei auf den gewünschten Bereich gesetzt sein. Die Routine setzt die Variablen PAGEX und MAPX, welche das aktive Speichergebiet angeben.

WO MODULE ABGELEGT WERDEN KÖNNEN:

Module können in RAM 0 im Bereich $2100–$FEFF abgelegt werden. Wenn die Startadresse unter $C000 liegt, wird jedoch der Benutzerspeicherbereich eingeschränkt. Der Bereich $AC00–$BFFF darf nicht verwendet werden, wenn das FONT-Paket für den 40-Zeichen-Bildschirm genutzt wird.

WO MODUL-VARIABLEN ABGELEGT WERDEN KÖNNEN:

Variablen, die von Aufruf zu Aufruf erhalten bleiben sollen, können entweder im Modul selbst oder im Bereich $1200–$12FF gespeichert werden. Zusätzlich können Tape- und RS-232-Puffer verwendet werden – vorausgesetzt, sie sind nicht anderweitig in Gebrauch. Imn der Zero Page ist nur die Speicherzelle $FF frei verfügbar. (Die Adressen $4C und $56 sind in der C-64-Dokumentation fälschlich als frei angegeben.)
Die Adressen der Zero-Page-Variablen (INF1 … TXTHI) sind unverändert. Die Adressen von COPY2 und COPY3 sind sowohl im C-64 als auch im C-128 gleich: COPY2: $0047–$0048, COPY3: $0049–$004A.
Die Variable RANGES wurde verschoben nach $1768–$1787 und TXT nach $09A1–$09F0.

PAKETBEISPIEL:

Das Paketbeispiel auf der C-128-Demodiskette wurde wie nachfolgend beschrieben geändert:
Das heißt:

```
    .lib C64symb
    .opt list           ; dieses Modul auflisten
;
    *=$8009             ;Startadresse
```

wurde geändert zu:

```
    .lib C128symb
    .opt list           ; dieses Modul auflisten
;
    *=$C000             ;Startadresse
```

Wie aus den obigen Ausführungen hervorgeht, wird es in den meisten Fällen leicht sein, ein C-64-Paket so anzupassen, dass es auch in einem C-128 verwendet werden kann.

LITERATUR:
Weitere Informationen zu COMAL-Paketen finden Sie im hervorragenden Buch:
Jesse Knight, COMAL 2.0 Packages, herausgegeben von der COMAL Users Group,
5501 Groveland Terrace, Madison, WI 53716-3251 USA.

Das Buch ist in Dänemark erhältlich bei:
COMAL TODAY DK
Postfach 122
DK-2300 Kopenhagen S

Anhang D: C128SYMB FOR COMMODORE 128 COMAL-80 REV. 2.02

```
;
;
;
; +-----------------------------------------------+
; ! !
; ! ****** COMMODORE 128 COMAL-80 ****** !
; ! !
; ! (C) COPYRIGHT UNICOMAL A/S 1984, 1986 !
; ! ALL RIGHTS RESERVED. !
; ! !
; ! !
; ! C128SYMB1 FOR COMMODORE 128 COMAL-80 REV. 2.02 !
; ! !
; +-----------------------------------------------+
;
;
;
TRUE =1
FALSE =0
;
;FYSISK MAP
;
FMAP1 = %00111111              RAM 0
FMAP2 = %00111110              RAM 0 + I/O
FMAP3 = %01111111              RAM 1
FMAP4 = %01111110              RAM 1 + I/O
;
FMAP5 = %00101111              EXT HI + RAM 0
FMAP6 = %00101110              EXT HI + RAM 0 + I/O
FMAP7 = %01101111              EXT HI + RAM 1
FMAP8 = %01101110              EXT HI + RAM 1 + I/O
;
FMAP9 = %10101111              EXT HI + RAM 2
FMAP10 = %10101110             EXT HI + RAM 2 + I/O
FMAP11 = %11101111             EXT HI + RAM 3
FMAP12 = %11101110             EXT HI + RAM 3 + I/O
;
```

```
FMAP13 = %00001110          KERNAL + RAM 0 + I/O
FMAP14 = %00001111          KERNAL + RAM 0 + CHARROM
FMAP15 = %00000000          KERNAL + RAM 0 + I/O+BASIC+MONITOR
;
COFMAP = FMAP5
POKMAP = 2                  MAP FOR POKE,PEEK,SYS
CPAGE = 5                   NORMAL COMAL MAP
GPAGE = 6                   NORMAL GRAFIK MAP
MPAGE = 15                  NORMAL MONITOR MAP
GRFMAP =FMAP6               FYSISK GRAFIK MAP
IQFMAP =FMAP13              FYSISK IRQ MAP RAM0+KER+IO
;
KEYPAG = 3                  MAP FOR FUNC.KEYS (RAM1)
TERPAG = 3                  MAP FOR TERMINATORS (RAM1)
DEFLEN = 32                 MAX 32 CHARS. PER KEY
KYBASE = $FD00              PLACE FOR DEFINITIONS
TRBASE = $FC00              PLACE FOR TERMINATORS
INDPAG = 3                  MAP FOR INDENT TABLE
INDTBL = $FB00              TABLE FOR INDENT VALUES (LIST)
BUF0 = $F900                BUFFER FOR CHANGE+SHOWKEYS
BUF1 = $FA00                BUFFER FOR CHANGE
BFFMAP = FMAP3              FYSISK MAP FOR BUF0+BUF1
TXMAP = $F700               TX MAP TABEL FOR ASCII CONV
RXMAP = $F800               RX MAP TABEL FOR ASCII CONV
MAFMAP = FMAP3              FYSISK MAP FOR MAP TABELS
;
IMMUA =FMAP13               INIT MMUPCRA KERNAL+RAM0+I/O
IMMUB =FMAP7                INIT MMUPCRB ENABLE CART+RAM1
IMMUC =FMAP6                INIT MMUPCRC ENABLE CART+I/O+RAM0
IMMUD =FMAP5                INIT MMUPCRD ENABLE CART+RAM0
;
PAGE1 =%10000000            CARTRIDGE OVERLAY 1 ($C000-$FFFF)
PAGE2 =%10010000            CARTRIDGE OVERLAY 2 ($C000-$FFFF)
PAGE3 =%10100000            CARTRIDGE OVERLAY 3 ($C000-$FFFF)
PAGE4 =%10110000            CARTRIDGE OVERLAY 4 ($C000-$FFFF)
PAGE5 =%11000000            CARTRIDGE OVERLAY 5 ($C000-$FFFF)
PAGE6 =%11010000            CARTRIDGE OVERLAY 6 ($C000-$FFFF)
PAGE7 =%11100000            CARTRIDGE OVERLAY 7 ($C000-$FFFF)
PAGE8 =%11110000            CARTRIDGE OVERLAY 8 ($C000-$FFFF)
;
```

```
        ;
        PAGEA =PAGE2+CPAGE          COMAL PAGE A
        PAGEB =PAGE5+CPAGE          COMAL PAGE B
        PAGEC =PAGE1+CPAGE          COMAL PAGE C
        PAGED =PAGE6+CPAGE          COMAL PAGE D
        PAGEE =PAGE7+CPAGE          COMAL PAGE E
        PAGEF =PAGE8+CPAGE          COMAL PAGE F
        ;
        PAGEGD =PAGE6+GPAGE         GRAFIK PAGE D
        PAGEGE =PAGE7+GPAGE         GRAFIK PAGE E
        PAGEGF =PAGE8+GPAGE         GRAFIK PAGE F
        ;
        ;
        PAGEKE = 13+PAGE1           NORMAL KERNAL MAP
        ;
        ;
        ; PAGE ZERO VARIABLES:
        ; ====================
        ;
        *=$0000
        ;
$0000   D6510 *=*+1                 6510 ON-CHIP DATA-DIRECTION REGISTER
$0001   R6510 *=*+1                 6510 ON-CHIP 6-BIT I/O/MAP-REGISTER:
        ;
$0002   BANK *=*+1                  MONITOR & LONG CALL/JUMP
$0003   PCHI *=*+1
$0004   PCLO *=*+1
$0005   SREG *=*+1
$0006   AREG *=*+1
$0007   XREG *=*+1
$0008   YREG *=*+1
$0009   STKPTR *=*+1
        ;
        ASAVE =AREG                 SAVE FOR .A (CALL/GOTO)
        XSAVE =XREG                 SAVE FOR .X (CALL/GOTO)
        PSAVE =SREG                 SAVE FOR .P (CALL/GOTO)
        GRWK3 =PCHI                 GRAPHIC WORK
        ;
        ;MEMORY MAP CONTROL
        ;
```

```
$000A   PAGEPT *=*+2            POINTER USED BY LOAD/STORE/EXEC
$000C   PAGEY *=*+1             OVERLAY USED FOR CONTROL OF JUMP TABLE
$000D   PMMUCR *=*+1            OLD MMUCR USED FOR CONTROL OF JUMP TABLE
        ;
        ; COMAL VARIABLES
        ;
$000E   LOCLPT *=*+2            CHAIN OF OLD VARIABLE DESCRIPTIONS
$0010   FORPT *=*+2             STACK ENTRY CHAIN
$0012   SCTYPE *=*+1            TYPE OF SYMBOL FROM SCANNER
$0013   TANSGN *=*+1            TAN SIGN / COMPARISON EVALUATION FLAG
$0014   CODE *=*+1              USED TO HOLD A GENERATED CODE
$0015   CPNT *=*+1              POINTER TO CODE BUFFER, CDBUF
        CLIMIT =255             LIMIT OF CPNT
$0016   SPROG *=*+2            PNT TO START OF PROGRAM
$0018   SVARS *=*+2             PNT TO START OF VARIABLE TABLE
$001A   SSTACK *=*+2            PNT TO START OF STACK
$001C   SMAX *=*+2              PNT TO TOP OF MEMORY
$001E   EXINF *=*+1             INF FOR RESULT EXPRESSION FROM EXPR
        SCERR =$01
        SCLNO =$08              SET, IF NUMBER CAN BE A LINE NUMBER
$001F   LNLEN *=*+1             LENGTH OF LINE TO BE EXECUTED
$0020   NPNT *=*+1              POINTER TO NAME
$0021   TPNT *=*+1              POINTER TO STRING
        TLIMIT =80
        ;
$0022   INDEX1 *=*+2            UTILITY POINTER
$0024   INDEX2 *=*+2            UTILITY POINTER
        INDEX =INDEX1
        ;
$0026   RESM1 *=*+1             PRODUCT AREA FOR MULTIPLICATION
$0027   RESM2 *=*+1
$0028   RESM3 *=*+1
$0029   RESM4 *=*+1
$002A   RESM5 *=*+1
        ;
        MULT1 =RESM1            FIRST OPERAND (MULT)
        MULT2 =RESM3            SECOND OPERAND (MULT)
        ;
        MOVELN =MULT1
        MOVETY =MULT2
```

```
        ;
$002B   DATAPT *=*+2            CURRENT DATA POINTER
$002D   STOS *=*+2              PNT TO TOP OF STACK
$002F   SFREE *=*+2             PNT TO FREE AREA OF VAR.RES
$0031   PRGPNT *=*+2            PNT TO START OF LINE
$0033   CODPNT *=*+1            PNT TO CODE DURING EXECUTION
$0034   SCLSD1 *=*+2            OLD SFREE (CLOSED)
$0036   SCLSD2 *=*+2            OLD STOS (CLOSED)
$0038   INF1 *=*+1
$0039   INF2 *=*+1              USED FOR OPERAND CHECKING
$003A   INF3 *=*+1
$003B   Q1 *=*+2                SHORT SPAN WORK AREAS
$003D   Q2 *=*+2
$003F   Q3 *=*+2
$0041   Q4 *=*+2
$0043   Q5 *=*+2
$0045   COPY1 *=*+2             WORK FOR COPY: FROM
$0047   COPY2 *=*+2             TO
        FDECPT =COPY2           CURRENT VARIABLE ADDRESS (DEC POINTER)
$0049   COPY3 *=*+2             LENGTH
$004B   BUS *=*+1               BUS=0: BUS IDLE; BUS<>0: BUS ACTIVE
$004C   STINF *=*+1             INFORMATION FOR STATEMENT:
        CMND =$01               NO LINE NUMBER
        TWOST =$02              ANOTHER STATEMENT FOLLOWS
        WHLDO =$04              AFTER WHILE ... DO
        FORDO =$08              AFTER FOR ... DO
        COMMNT =$10             STATEMENT ENDED BY COMMENT
        IFTHEN =$20             AFTER IF ... THEN
        REPUNT =$40             AFTER REPEAT ... UNTIL
        ;
$004D   EXCINF *=*+1            EXECUTION INFORMATION:
        ESCTRP =$02             ESCAPE IS TRAPPED (STOP)
        SRQ =$04                MAKE CALL OF COMAL INTERRUPT HANDLER.
        ESCMET =$08             ESCAPE MET (STOP)
        SRQON =$10              SRQ ENABLED
        USRQON =$20             USER REQUEST ENABLED
        SFTSRQ =$80             SOFTWARE SRQ ONLY
        ;
        ; VARIABLES FOR FLOATING POINT PACKAGES
        ;
```

```
$004E   TEMPF3 *=*+6            MISC. FP WORK AREA
        NUMBER =TEMPF3          FP WORK AREA (SYNTAX ANALYSIS)
$0054   ESCAPE *=*+1            STOP KEY FLAG
$0055   INTEGR *=*+1            FP WORK
$0056   OLDOV *=*+1             OLD OVERFLOW (ROUNDING)
$0057   TEMPF1 *=*+5            MISC. FP WORK AREA (5 BYTES)
$005C   TEMPF2 *=*+5            MISC. FP WORK AREA (5 BYTES)
        DECCNT =TEMPF2+1
        TENEXP =TEMPF2+2
        ;
$0061   AC1 *=*+6               ACCUM#1:
        AC1E =AC1+0             EXPONENT
        AC1M1 =AC1+1            MANTISSA 1
        AC1M2 =AC1+2            MANTISSA 2
        AC1M3 =AC1+3            MANTISSA 3
        AC1M4 =AC1+4            MANTISSA 4
        AC1S =AC1+5             SIGN
        ;
$0067   DEGREE *=*+1            SERIES EVALUATION CONSTANT POINTER
$0068   BITS *=*+1              ACCUM#1: HI-ORDER (OVERFLOW)
        ;
$0069   AC2 *=*+6               ACCUM#2:
        AC2E =AC2+0             EXPONENT
        AC2M1 =AC2+1            MANTISSA 1
        AC2M2 =AC2+2            MANTISSA 2
        AC2M3 =AC2+3            MANTISSA 3
        AC2M4 =AC2+4            MANTISSA 4
        AC2S =AC2+5             SIGN
        ;
$006F   ARISGN *=*+1            SIGN COMPARISON, ACC#1 VS ACC#2
$0070   FACOV *=*+1             ACCUM#1: LO-ORDER (ROUNDING)
$0071   POLYPT *=*+2            POINTER TO POLYNOM.
        FBUFPT =POLYPT
        ;
        ; MORE COMAL VARIABLES
        ;
$0073   PRPROC *=*+3            CHAIN OF LOCAL NAMES (PREPASS)
$0076   INDPNT *=*+1            POINTER TO LAST CODE WHERE AN ADDRESS WAS LOADED
$0077   SCFLAG *=*+1            FLAGS IN SCANNER:
$0078   LNNO *=*+2              LINE NUMBER
```

```
$007A   MOVEAD *=*+2           ADDRESS FOR MOVE
$007C   TXTLO *=*+1            ADDRESS OF TEXT FOR PRTXT
$007D   TXTHI *=*+1
$007E   XX *=*+2               CURRENT X (GRAPHICS)
$0080   YY *=*+2               CURRENT Y (GRAPHICS)
$0082   GRWK1 *=*+2
$0084   GRWK2 *=*+2
$0086   GCOL *=*+1             GRAPHIC DRAW COLOR
$0087   GBACK *=*+1            GRAPHIC BACKGROUND
$0088   EXCFLG *=*+1           FLAGS:
        NWNAME =$01            NEW NAME HAS BEEN INSERTED
        NWLINE =$02            NEW LINE HAS BEEN INSERTED
$0089   CHARPT *=*+1           PNT TO INBUF
$008A   CHAR *=*+1             CHAR FROM INCHAR
$008B   RNDX *=*+5             RANDOM NUMBER SEED
        ;
        ;VARIABLES FOR I/O
        ;
$0090   STATUS *=*+1           I/O OPERATION STATUS
$0091   STKEY *=*+1            STOP KEY FLAG
$0092   SVXT *=*+1             TEMPORARY
$0093   VERCK *=*+1            LOAD OR VERIFY FLAG
$0094   C3P0 *=*+1             IEEE BUFFERED CHAR FLAG
$0095   BSOUR *=*+1            CHAR BUFFER FOR IEEE
$0096   SYNO *=*+1             CASSETTE SYNC #
$0097   XSAV *=*+1             TEMP FOR BASIN
$0098   LDTND *=*+1            HOW MANY FILES OPEN
$0099   DFLTN *=*+1            DEFAULT INPUT DEVICE #
$009A   DFLTO *=*+1            DEFAULT OUTPUT DEVICE #
$009B   PRTY *=*+1             CASSETTE PARITY
        T3 =PRTY
$009C   DPSW *=*+1             CASSETTE DIPOLE SWITCH
$009D   MSGFLG *=*+1           OS MESSAGE FLAG
$009E   PTR1 *=*+1             CASSETTE ERROR PASS 1
        T1 =PTR1               TEMPORARY 1
$009F   PTR2 *=*+1             CASSETTE ERROR PASS 2
        TMPC =PTR2
        T2 =PTR2               TEMPORARY 2
$00A0   TIME *=*+3             24 HOUR CLOCK IN 1/60 SEC.
        R2D2 =*                SERIAL BUS USAGE
```

$00A3	PCNTR *=*+1	CASSETTE STUFF
	BSOUR1 =*	TEMP USED BY SERIAL ROUTINE
$00A4	FIRT *=*+1	
	COUNT =*	TEMP USED BY SERIAL ROUTINE
$00A5	CNTDN *=*+1	CASSETTE SYNC COUNTDOWN
$00A6	BUFPT *=*+1	TAPE BUFFER POINTER
	SHCNL =*	CASSETTE SHORT COUNT
$00A7	INBIT *=*+1	RS232 RCVR INPUT BIT STORAGE
	RER =*	CASSETTE READ ERROR
$00A8	BITCI *=*+1	RS232 RCVR BIT COUNT IN
	REZ =*	CASSETTE READING ZEROES
$00A9	RINONE *=*+1	RS232 RCVR FLAG FOR START BIT CHECK
	RDFLG =*	CASSETTE READ MODE
$00AA	RIDATA *=*+1	RS232 RCVR BYTE BUFFER
$00AB	RIPRTY *=*+1	RS232 RCVR PARITY STORAGE
	SHCNH =RIPRTY	CASSETTE SHORT COUNT
$00AC	SAL *=*+1	POINTER: TAPE BUFFER/SCREEN SCROLLING
$00AD	SAH *=*+1	
$00AE	EAL *=*+1	
$00AF	EAH *=*+1	
$00B0	CMP0 *=*+1	TAPE TIMING CONSTANT
$00B1	TEMP *=*+1	TAPE TIMING CONSTANT
$00B2	TAPE1 *=*+2	START OF TAPE BUFFER
	;	
	SNSW1 =*	CASSETTE STUFF
$00B4	BITTS *=*+1	RS232 TRNS BIT COUNT
$00B5	NXTBIT *=*+1	RS232 TRNS NEXT BIT TO BE SENT
	DIFF =NXTBIT	EOT RECEIVED FROM TAPE
	PRP =*	
$00B6	RODATA *=*+1	RS232 TRNS BYTE BUFFER
$00B7	FNLEN *=*+1	LENGTH OF CURRENT FILE NAME
$00B8	LA *=*+1	CURRENT FILE LOGICAL ADDRESS
$00B9	SA *=*+1	CURRENT FILE SECONDARY ADDRESS
$00BA	FA *=*+1	CURRENT FILE PRIMARY ADDRESS
$00BB	FILADR *=*+2	CURRENT FILE NAME ADDRESS
	OCHAR =*	
$00BD	ROPRTY *=*+1	RS232 TRNS PARITY BUFFER
$00BE	FSBLK *=*+1	CASSETTE READ BLOCK COUNT
$00BF	MYCH *=*+1	SERIAL WORD BUFFER
$00C0	CAS1 *=*+1	CASSETTE MANUAL/CONTROLLED SWITCH

```
$00C1   STAL *=*+1                      TAPE START ADDRESS
$00C2   STAH *=*+1
        TMP2 =*
$00C3   MEMUSS *=*+2                    TAPE LOAD TEMPS
$00C5   DATA *=*+1                      TAPE R/W DATA
$00C6   BA *=*+1                        BANK CUR LOAD/SAVE/VER OPE.
$00C7   FNBANK *=*+1                    BANK FOR CUR FILENAME
        ;
$00C8   RIBUF *=*+2                     RS-232 INPUT BUF PTR
$00CA   ROBUF *=*+2                     RS-232 OUTPUT BUF PTR
        ;
        ;40/80 COLUMN SCREEN EDITOR
        ;
        ;GLOBAL SCREEN EDITOR VAR
        ;
$00CC   KEYTAB *=*+2                    KEYSCAN TABLE PTR
$00CE   IMPARM *=*+2                    PRIMM UTIL. STRING PTR
$00D0   NDX *=*+1                       INDEX TO KEYB. QUEUE
$00D1   KYNDX *=*+1                     PENDING FUNC. KEY FLAG
$00D2   KEYIDX *=*+1                    INDEX INTO PEND. FUNC. KEY STRING
$00D3   SHFLAG *=*+1                    KEYSCAN SHIFT KEY STATUS
$00D4   SFDX *=*+1                      KEYSCAN CUR. KEY INDEX
$00D5   LSTX *=*+1                      KEYSCAN LAST KEY INDEX
        NEWKEY =LSTX
$00D6   CRSW *=*+1                      <CR> INPUT FLAG
        ;
$00D7   EMODE *=*+1                     40/80 COL. MODE FLAG
$00D8   GRAPHM *=*+1                    TEXT/GRAPHIC MODE FLAG
        ;
$00D9   CHAREN *=*+1                    RAM/ROM VIC CHR FETCH FLAG (BIT-2)
        ;
        ;THE FOLLOWING ARE SHARED BY SEVERAL EDIT. ROUTINES
        ;
$00DA   SEDSAL *=*+2                    POINTER FOR MOXLIN
$00DC   SEDEAL *=*+2
$00DE   SEDT1 *=*+1                     SAWPO
$00DF   SEDT2 *=*+1
        ;
        *=SEDSAL                        SHARE WITH SEDSAL
$00DA   KEYSIZ *=*+1                    PROGRAMMABLE KEY VAR.
```

```
$00DB   KEYLNG *=*+1
$00DC   KEYNUM *=*+1
$00DD   KEYNXT *=*+1
$00DE   KEYBNK *=*+1
$00DF   KEYTMP *=*+1
        ;
        *=SEDSAL                    SHARE WITH SEDSAL
$00DA   BITMSK *=*+1                TEMP FOR TAB & LINE WRAP ROUT.
$00DB   SAVER *=*+1
$00DC   *=*+4
        ;
        ;LOCAL SCREEN EDITOR VAR.
        ;
$00E0   PNT *=*+2                   PTR TO CUR LINE(TEXT)
        CURLIN =PNT
$00E2   USER *=*+2                  PTR TO CUR LINE(ATTRIBUTE)
        ;
$00E4   SCBOT *=*+1                 WINDOW LOWER LIMIT
$00E5   SCTOP *=*+1                 WINDOW UPPER LIMIT
$00E6   SCLF *=*+1                  WINDOW LEFT MARGIN
$00E7   SCRT *=*+1                  WINDOW RIGHT MARGIN
        ;
$00E8   LSXP *=*+1                  CUR INPUT COLUMN START
$00E9   LSTP *=*+1                  CUR INPUT LINE START
$00EA   INDX *=*+1                  CUR INPUT LINE END
        ;
$00EB   TBLX *=*+1                  CUR CURSOR LINE
        ROWPOS =TBLX
$00EC   PNTR *=*+1                  CUR CURSOR COLUMN
        COLPOS =PNTR
        ;
$00ED   LINES *=*+1                 MAX NUMBER OF SCREEN LINES
$00EE   COLUMN *=*+1                MAX NUMBER OF SCREEN COLUMNS
$00EF   DATAX *=*+1                 CUR CHARACTER TO PRINT
$00F0   LSTCHR *=*+1                PREV. CHARACTER PRINTED(FOR ESC)
$00F1   COLOR *=*+1                 CUR ATTRIBUTE TO PRINT(DEF FGND COLOR)
$00F2   TCOLOR *=*+1                SAVED ATRIBUTE TO PRINT
        ;
$00F3   RVS *=*+1                   REVERSE MODE FLAG
$00F4   QTSW *=*+1                  QUOTE MODE FLAG
```

```
        QUOTE =QTSW
$00F5   INSRT *=*+1             INSERT MODE FLAG
$00F6   INSFLG *=*+1            AUTO-INSERT MODE FLAG
        ;
$00F7   LOCKS *=*+1             DISABLE <C=><SHIFT>, <CRTL>-S
$00F8   SCROLL *=*+1            DISABLE SCREEN SCROLL, LINE LINKER
$00F9   BEEPER *=*+1            DISABLE <CRTL>-S
$00FA   *=*+1                   CHANGED BY SWAP EDIT MODE
        ;
        ;MORE COMAL VAR
        ;
$00FB   MAPX *=*+1              MAP FOR LOAD/STORE/EXEC ROUTINES
$00FC   PAGEX *=*+1             OVERLAY FOR LOAD/STORE/EXEC ROUTINES
        ;GRAPHIC VAR
$00FD   RESOL *=*+1             GRAPHICS RESOLUTION
$00FE   GCOLH *=*+1             GRAPHICS PENCOLOR*16
        ;
        ;END OF PAGE ZERO
        ;
        *=$0100
$0100   STACK *=*+256           SYSTEM STACK
        FBUFFR =STACK           FPASC WORK AREA (15 BYTES)
        BAD =STACK              TAPE INPUT ERROR LOG
        ;
        ;
$0200   ERTLEN *=*+1            LENGTH OF ERTEXT
        MAXERT =79              MAX. LENGTH OF ERTEXT
$0201   ERTEXT *=*+MAXERT       BUFFER TO HOLD ERROR MESSAGE
        ;
        FNMAX =55
$0250   FILNAM *=*+FNMAX        USED FOR STORAGE OF FILE NAME
        TBUFF =FILNAM           USED FOR STORAGE OF DISC COMMAND
        ;
        ;STORAGE FOR CON COMMAND:
        ;
$0287   CONPNT *=*+2            OLD PRGPNT
$0289   CONFLG *=*+1            OLD EXCINF
$028A   CONCOD *=*+1            OLD CODPNT
$028B   CONFOR *=*+2            OLD FORPT
        ;
```

```
$028D   FPWORK *=*+1
        ;
        ;
        *=$02A2
        ;
$02A2   FETCH *=*+13            LDA( - ),Y FROM ANY BANK
        FETVEC =FETCH+8
        ;
$02AF   STASH *=*+15            STA( - ),Y TO ANY BANK
        STAVEC =STASH+10
        ;
$02BE   CMPARE *=*+15           CMP( - ),Y TO ANY BANK
        CMPVEC =CMPARE+10
        ;
$02CD   JSRFAR *=*+22           JSR XXXX TO ANY BANK & RETURN
        ;
$02E3   JMPFAR *=*+25           JMP XXXX TO ANY BANK
        ;
        ;*=$02FC
        ;
        ;
        ;COMAL VECTORS
        ;
$02FC   IOVECT =*               VECTORS:
$02FC   TRAPVC *=*+2             PAGEB; ERROR HANDLER
$02FE   EXTNVC *=*+2             PAGEB; EXTERNAL LOAD
$0300   USRQVC *=*+2             PAGEB; INTERRUPT FACILITY
$0302   IERTXT *=*+3             ERROR MESSAGE DATA
$0305   IGETLN *=*+2             PAGEA; INPUT COMMAND LINE
$0307   ISAVEC *=*+2             PAGEC; SAVE ADDITIONAL INFO
$0309   ILOADC *=*+2             PAGEC; LOAD ADDITIONAL INFO
$030B   IFNKEY *=*+2             PAGEA; HANDLE FUNCTION KEYS
        ;
        *=$0314                 KERNAL INDIRECT VECTORS
        ;
        ;KERNAL VECTORS:
        ;
$0314   IIRQ *=*+2              IRQ RAM VECTOR
        CINV =IIRQ
        IRQVCT =IIRQ
```

```
        ;
$0316   IBRK *=*+2                  BRK INSTR RAM VECTOR
        CBINV =IBRK
        BRKVCT =IBRK
        ;
$0318   INMI *=*+2                  NMI RAM VECTOR
        NMIVCT =INMI
        ;
$031A   IOPEN *=*+2                 OPEN ROUTINE VECTOR
$031C   ICLOSE *=*+2                CLOSE ROUTINE VECTOR
$031E   ICHKIN *=*+2                CHKIN ROUTINE VECTOR
$0320   ICKOUT *=*+2                CKOUT ROUTINE VECTOR
$0322   ICLRCH *=*+2                CLRCHN ROUTINE VECTOR
$0324   IBASIN *=*+2                CHRIN ROUTINE VECTOR
$0326   IBSOUT *=*+2                CHROUT ROUTINE VECTOR
$0328   ISTOP *=*+2                 STOP ROUTINE VECTOR
$032A   IGETIN *=*+2                GETIN ROUTINE VECTOR
$032C   ICLALL *=*+2                CLALL ROUTINE VECTOR
$032E   EXMON *=*+2                 FOR MACHINE LANGUAGE MONITOR
$0330   ILOAD *=*+2                 LOAD ROUTINE VECTOR
$0332   ISAVE *=*+2                 SAVE ROUTINE VECTOR
        ;
        ;EDITOR VECTORS
        ;
$0334   CTLVEC *=*+2                EDITOR: PRINT 'CONTRL' INDIRECT
$0336   SHFVEC *=*+2                EDITOR: PRINT 'SHIFTD' INDIRECT
$0338   ESCVEC *=*+2                EDITOR: PRINT 'ESCAPE' INDIRECT
$033A   KEYVEC *=*+2                EDITOR: KEYSCAN LOGIC INDIRECT
$033C   KEYCHK *=*+2                EDITOR: STORE KEY INDIRECT
        ;
$033E   DECODE *=*+12               VECTORS TO KEYB. MATRIX DEC TABLES
        ;
        *=$034A
$034A   KEYD *=*+10                 IRQ KEYBOARD BUFFER
        KEYBUF = KEYD
        ;
$0354   TABMAP *=*+10               BITMAP OF TAB STOPS
$035E   BITABL *=*+4                BITMAP OF LINE WRAPS
        ;
$0362   LAT *=*+10                  TABLE OF LA'S
```

```
$036C   FAT *=*+10                      TABLE OF FA'S
$0376   SAT *=*+10                      TABLE OF SA'S
        ;
        ;COMAL:
        ;OPEN FILE MODES (COPEN & CFNAME):
        ;
        MREAD =1                        READ
        MWRITE =2                       WRITE
        MREL =4                         RELATIVE (RANDOM)
        ;
        ;FILE ATTRIBUTES (COPEN):
        ;
        MDISK =8                        DISK FILE
        MS:=16                          SECONDARY ADDRESS SELECTION
        ML =32                          AUTO LINEFEED
        MT =64                          TIME OUT
        MA =128                         ASCII I/O
        ;
        ;FILE TYPES (CFNAME):
        ;
        TAPPND =8
        TPRG =16
        TSEQ =TPRG+TPRG
        TREL =TSEQ+TPRG
        TUSR =TREL+TPRG
        ;
$0380   MODET *=*+10                    OPEN MODE FOR FILES
$038A   COUNTT *=*+10                   TABLE OF BYTE COUNT FOR FILES
$0394   STT *=*+10                      STATUS FOR OPENED FILES
$039E   RECOTL *=*+10                   TABLE OF RECORD POS. FOR FILES
$03A8   RECOTH *=*+10
        ;
$03B2   LIBPT *=*+1                     PTR TO PLACE FOR NEXT LIBRARY DESCRIPTION
        ;
        LIBLEN =10                      NO. OF LIBRARY DESCRIPTIONS
        ;
$03B3   LIBLO *=*+LIBLEN
$03BD   LIBHI *=*+LIBLEN
$03C7   LIBPAG *=*+LIBLEN
        ;
```

```
$03D1   KEYLEN *=*+16               LENGTHS OF FUNC.KEY DEF'S
$03E1   KLEN *=*+1                  # OF CHARS LEFT OF DEF.
$03E2   KPNT *=*+2                  PNT TO KEY DEF
        ;
        ; BASIC/KERNAL DMA REQ. RAM CODE
        ;
        *=$0400                     VIDEO MATRIX #1: VIC 40-COLUMN TEXT
        VICSCN *=*+1024
        ;
        *=VICSCN
        SCSIZE =1000                SCREEN SIZE
$0400   SCREEN *=*+SCSIZE           SCREEN MEMORY AREA
        *=*+16                      SCREEN MEMORY EXCESS
$07F8   SPRPNT *=*+8                SPRITE DATA POINTERS
        ;
        ;COMAL VAR
        ;
        ;
        INLEN =162
$0800   INBUF *=*+INLEN             INPUT BUFFER
$08A2   CDBUF *=*+CLIMIT            CODE BUFFER ; 255
$09A1   TXT *=*+TLIMIT              STRING CONSTANT BUFFER ;80
        TXT1 =TXT-1
        ;
$09F1   FLEVEL *=*+1                FOR/TRAP NESTING LEVEL DURING PREPASS
$09F2   Q6 *=*+2                    TEMPORARY
$09F4   Q7 *=*+2                    TEMPORARY
$09F6   Q8 *=*+2                    TEMPORARY
$09F8   Q9 *=*+2                    TEMPORARY
$09FA   INDPTR *=*+1                POINTER TO INDTBL
$09FB   CURCHK *=*+1                SAVE FOR FILE# IN CHKIN,CKOUT
$09FC   EXPSTA *=*+1                EXPECTED DISK STATUS
        ;
$09FD   OPTIO1 *=*+1                OPTION1: =0: , =1 ; ZONE SEPARATOR
$09FE   DRSCMD *=*+1                DEFAULT RS232 CMD REG
$09FF   DRSCTR *=*+1                DEFAULT RS232 CTRL REG
        .END

        ;
        ;
```

```
          ;
          ; +-----------------------------------------------+
          ; ! !
          ; ! ****** COMMODORE 128 COMAL-80 ****** !
          ; ! !
          ; ! (C) COPYRIGHT UNICOMAL A/S 1984, 1986 !
          ; ! ALL RIGHTS RESERVED. !
          ; ! !
          ; ! !
          ; ! C128SYMB2 FOR COMMODORE 128 COMAL-80 REV. 2.02 !
          ; ! !
          ; +-----------------------------------------------+
          ;
          ;
          ;
          ;ABSOLUT KERNAL VARIABLES
          ;
          ;
$0A00     SYSVEC *=*+2              VECTOR TO RESTART SYSTEM
$0A02     DEJAVU *=*+1              KERNAL WARM/COLD INIT. STATUS BYTE
$0A03     PALNTS *=*+1              PAL/NTSC SYSTEM FLAG
$0A04     ISTAT *=*+1               FLAGS RESET VS. NMI STATUS FOR INIT ROUT
$0A05     MEMSTR *=*+2              START OF MEMORY
$0A07     MEMSIZ *=*+2              TOP OF MEMORY
          ;
$0A09     IRQTMP *=*+2              TAPE HANDLER IRQ INDIRECT
$0A0B     CASTON *=*+1              TAPE SENSE DURING TAPE OP.
$0A0C     KIKA26 *=*+1              TAPE READ TEMP
$0A0D     STUPID *=*+1              TAPE READ D1 IRQ IND.
          ;
$0A0E     TIMOUT *=*+1              FAST SERIAL TIMEOUT FLAG
          ;
          ;
          ;RS232 STORAGE
          ;
$0A0F     ENABL *=*+1               RS-232 ENABLES
$0A10     M51CTR *=*+1              6551 CONTROL REGISTER IMAGE
$0A11     M51CDR *=*+1              6551 COMMAND REGISTER IMAGE
$0A12     M51AJB *=*+2              NON-STANDARD BPS (TIME/2-100) USA
$0A14     RSSTAT *=*+1              6551 STATUS REGISTER
```

```
$0A15   BITNUM *=*+1                    NUMBER OF BITS LEFT TO SEND
$0A16   BAUDOF *=*+2                    BAUD RATE: FULL BIT TIME (MICROSEC)
$0A18   RIDBE *=*+1                     INDEX TO END OF INPUT BUFFER
$0A19   RIDBS *=*+1                     START OF INPUT BUFFER (PAGE)
$0A1A   RODBS *=*+1                     START OF OUTPUT BUFFER (PAGE)
$0A1B   RODBE *=*+1                     INDEX TO END OF OUTPUT BUFFER
        ;
$0A1C   SERIAL *=*+1                    FAST SERIAL INT/EXT FLAG
        ;
$0A1D   TIMER *=*+3                     DEC. JIFFLE REGISTER
        ;
        ;GLOBAL SCREEN EDITOR STORAGE
        ;
$0A20   KBFLIM *=*+1                    KEYBOARD QUEUE MAX SIZE
$0A21   PAUSE *=*+1                     <CRTL>-S FLAG
$0A22   RPTFLG *=*+1                    ENABLE KEY REPEAT
$0A23   KOUNT *=*+1                     DELAY BETWEEN KEY REPEATS
$0A24   DELAY *=*+1                     DELAY BEFORE KEY REPEAT
$0A25   LSTSHF *=*+1                    DELAY BETWEEN <C=><SHFT> TOGGLES
        ;
$0A26   BLNON *=*+1                     VIC CURSOR MODE
$0A27   BLNSW *=*+1                     VIC CURSOR DISABLE
$0A28   BLNCT *=*+1                     VIC CURSOR BLINK COUNTER
$0A29   GDBLN *=*+1                     VIC CURSOR CHAR BEFORE BLINK
$0A2A   GDCOL *=*+1                     VIC CURSOR COLOR BEF. BLINK
        ;
$0A2B   CURMOD *=*+1                    VDC CURSOR MODE (WHEN ENABLED)
        ;
$0A2C   VM1 *=*+1                       VIC TEXT SCREEN.CHAR BASE POINTER
$0A2D   VM2 *=*+1                       VIC BIT-MAP BASE POINTER
$0A2E   VM3 *=*+1                       VDC TEXT SCREEN BASE
$0A2F   VM4 *=*+1                       VDC ATTRIBUTE BASE
        ;
$0A30   LINTMP *=*+1                    TEMP PTR TO LAST LINE FOR LOOP4
$0A31   SAV80A *=*+1                    TEMP FOR 80-COL ROUTINES
$0A32   SAV80B *=*+1
$0A33   CURCOL *=*+1                    VDC CURSOR COLOR BEFORE BLINK
$0A34   SPLITK *=*+1                    VIC SPLIT SCREEN RASTER VALUE
$0A35   FNADRX *=*+1                    SAVE .X DURING BANK OPERATION
$0A36   PALCNT *=*+1                    COUNTER FOR PAL SYSTEMS
```

```
$0A37   SPEED *=*+1                     SAVE SYSTEM SPEED DUR. TAPE/SERIAL OP.
$0A38   SPRITE *=*+1                    SAVE SPRITE ENABLES DUR. TAPE OP.
$0A39   BLNING *=*+1                    SAVE BLANKING STATUS DUR. TAPE OP.
$0A3A   HLDOFF *=*+1                    FLAG FOR USER FULL CONTROL OF VIC
$0A3B   LDTBSA *=*+1                    HIGH BYTE OF SA OF VIC SCREEN
        HIBASE =LDTBSA                  64-KERNAL NAME HIBASE
$0A3C   *=*+2                           8563 BLOCK FILL KLUDGE
        ;
        ;LOCAL SCREEN EDITOR VAR. SWAP AREA
        ;
        ;
        *=$0A40                         LOCAL VARIABLES
        ;
        *=$0A60                         LOCAL TAB MAP & WRAP TABLES
        ;
        ;MONITOR ABSOLUTE DECLARATIONS
        ;
        *=$0A80
        ;
        ;FUNCTION KEY ROM CARD TABLES
        ;
        *=$0AC0
$0AC0   CURBNK *=*+1                    CURRENT FUNC.KEY ROM BANK
$0AC1   PAT *=*+4                       PHYSICAL ADDRESS TABLE (ID OF LOGGED-IN CARD)
$0AC5   DKFLAG *=*+1                    RESERVED FOR FOREIGN SCREEN EDITORS
        ;
        *=$0B00
        BUFSZ =192                      TAPE BUFFER SIZE
        TBUFFR *=*+BUFSZ                CASETTE BUFFER
        ;
        *=$0C00                         RS-232 INPUT BUFFER
        RS232I
        ;
        *=$0D00                         RS-232 OUTPUT BUFFER
        RS232O
        ;
        *=$0E00                         SPRITE DEF. AREA (MUST BE BELOW $1000 !!!)
        ;
        ;*=$1000 ;PROGRAMMABLE FUNCTION KEY DEF.
        ;PKYNUM =10 ;NUM.OF DEFINABLE KEYS (F1-F8,<SHIFT>RUN,HELP)
```

```
$1000   ;PKYBUF *=*+PKYNUM ;PROG. FUNC. KEY LENGTHS TABLE
$100A   ;PKYDEF *=*+256-PKYNUM ;PROG. FUNC. KEY STRING
        ;
        *=$1100                 CP/M RESET CODE,BASIC DOS & VSP VAR.
        ;
        ;COMAL RAM CODE
        ;
        KJMPTB =*
        ;
        ;KERNAL JUMPTABLE
        ;
$1100   CINT *=*+9              KERNAL: CINT
$1109   RESTOR *=*+9            KERNAL: RESTOR
$1112   RAMTAS *=*+9            KERNAL: RAMTAS
$111B   IOINIT *=*+9            KERNAL: IOINIT
$1124   CHKIN *=*+9             KERNAL: CHKIN
$112D   CKOUT *=*+22            KERNAL: CKOUT
$1143   CLRCH *=*+23            KERNAL: CLRCH
$115A   OPEN *=*+9              KERNAL: OPEN
$1163   CLOSE *=*+9             KERNAL: CLOSE
$116C   RDT *=*+9               KERNAL: RDT
$1175   WRT *=*+9               KERNAL: WRT
$117E   GET *=*+9               KERNAL: GET
$1187   SETNAM *=*+9            KERNAL: SETNAM
$1190   PLOT *=*+9              KERNAL: PLOT
$1199   DISPLY *=*+9            EDITOR: DISPLY
$11A2   CURSOR *=*+9            EDITOR: CURSOR
$11AB   ESCFNC *=*+9            EDITOR: ESCFNC
$11B4   SPIOUT *=*+9            KERNAL: SPIOUT
$11BD   SWAP *=*+9              EDITOR: SWAPPER
$11C6   WINDOW *=*+9            EDITOR: WINDOW
$11CF   PRINT *=*+9             EDITOR: PRINT
        ;
        *=$1200                 UNUSED
        ;
        *=$1300                 UNALLOCATED RAM
        ;
$1300   STDPK1 *=*+150          GRAPHIC IRQ RAM CODE
        ;
$1396   STDPK2 *=*+900
```

```
        ;
$171A   SPSAV *=*+1             SAVE OF .S DURING EXECUTION
$171B   SCINF *=*+2             NAME POINTER
        ;
        PSTART =*
        ;
$171D   AUTO1 *=*+2             CURRENT LINE NUMBER FOR AUTO
$171F   AUTOST *=*+2            STEP FOR AUTO
        ;
$1721   DSTART *=*+2            START OF DATA QUEUE
        ;
$1723   TABSET *=*+1            VALUE OF LAST ZONE STATEMENT
$1724   ALTPOS *=*+1            POSITION IN SELECT OUTPUT FILE
$1725   INTRNO *=*+2            PROCEDURE GIVEN BY INTERRUPT STMT.
        ;
        PEND =*
        ;
$1727   ERRPNT *=*+1            CHAR POS OF ERROR
$1728   NORINT *=*+2            NORMAL INTERRUPT VECTOR
$172A   SAFE *=*+1              SAFE STATUS
$172B   MAINRV *=*+1            MAIN REVISION
$172C   SUBRV *=*+1             SUB REVISION
$172D   TESTRV *=*+1            TEST VERSION
$172E   MSGLIN *=*+2            ADDRESS OF MESSAGE LINE
$1730   MSGPRE *=*+1            MESSAGE PRESENT FLAG
$1731   OLDPNT *=*+2            SAVE PNT
$1733   UPPER2 *=*+1            COPY OF BORGE
$1734   EXTPRC *=*+1            FLAG FOR LOADING OF EXTERNAL PROC/FUNC
$1735   EOFBLK *=*+1            BURST READ LAST BLOK
$1736   LOADMD *=*+1            LOAD MODE: =0 BURST =1 NORMAL LOAD
$1737   EXTCNT *=*+2            NESTING LEVEL OF EXTERNAL PROC/FUNC
$1739   INSMOD *=*+1            SAVE LOC FOR INSFLG
$173A   LUNIT *=*+1             LAST DISC (STATUS)
$173B   BORGE *=*+1             SPECIAL FLAGS FOR LISTING
$173C   OPENFL *=*+1            FLAG USED BY COPEN
$173D   DFUNLN *=*+1            LENGTH OF DEFAULT UNIT TEXT
$173E   DFUNIT *=*+2            DEFAULT UNIT (POWER UP VALUE: .BYTE '0')
$1740   DEFOUT *=*+1            SELECT OUTPUT FLAG
        ;
        ;
```

```
$1741   PPAGE *=*+1                  OVERLAY TO PEEK/POKE/SYS
$1742   LOADIN *=*+1                 <>0: LOADING COMAL PROGRAM
$1743   UNITFL *=*+1                 0: SIMP.DEV; 1: DRIVE; 2: CASSETTE
$1744   MODE *=*+1                   FILE MODE
$1745   CSTAT *=*+1                  STATUS OF COMAL PROGRAM
        ;
        ;1: INPUT ANALYSIS FROM SCREEN
        ;2: - - - FILE
        ;3: PREPASSING
        ;4: EXECUTING A COMMAND
        ;5: EXECUTING PROGRAM
        ;
$1746   LSTFLG *=*+1                 BIT VECTOR FOR RCREAT:
        ;
        ININD =$01
        OUTIND =$02
        LSTCON =$04
        NOIND =$08
        LSTEXT =$10
        ;
$1747   LPMODE *=*+1                 DEFAULT PRINTER OPEN MODE
$1748   LPSA *=*+1                   DEFAULT PRINTER SECONDARY ADDRESS
$1749   LPFA *=*+1                   DEFAULT PRINTER UNIT
$174A   SPMODE *=*+1                 DEFAULT RS232 OPEN MODE
$174B   SPSA *=*+1                   DEFAULT RS232 SECONDARY ADDRESS
$174C   SPFA *=*+1                   DEFAULT RS232 UNIT
        ;
$174D   RECDEL *=*+1                 RECORD POSITIONING DELAY
$174E   ENDADR *=*+2                 TOP OF RAM
$1750   HEADLN *=*+1                 POWER ON MESSAGE FLAG
        ;
$1751   KWTAB *=*+2                  KEYWORD TABLE (PAGEA)
        ;
$1753   DFBORD *=*+1                 DEFAULT BORDER COLOR
$1754   DFBACK *=*+1                 DEFAULT BACKGROUND COLOR
$1755   DFFORG *=*+1                 DEFAULT FOREGROUND COLOR
$1756   ACBORD *=*+1                 ACTUAL TEXT BORDER
$1757   ACBACK *=*+1                 ACTUAL TEXT BACKGROUND
$1758   DF8FOR *=*+1                 DEFAULT 80 CHAR FOREGROUND COLOR
$1759   DF8BCK *=*+1                 DEFAULT 80 CHAR BACKGROUND COLOR
```

```
        ;
$175A   DEFINP *=*+1                SELECT INPUT FLAG
$175B   HZ50 *=*+1                  0=60 HZ, 1=50 HZ TOD
$175C   MSKPAG *=*+1                MASKED PAGE INFO
$175D   INPOFS *=*+1                INPUT OFFSET
$175E   TRCHAR *=*+1                TERMINATOR CHAR
$175F   TEROFS *=*+1                TERMINATOR OFFSET
$1760   CURMOF *=*+1                VDC CURSOR MODE WHEN DISABLET
$1761   CPUSPD *=*+1                CPU SPEED (0=AUTO,$80=1MHZ,$C0=2MHZ)
$1762   CN2MHZ *=*+1
$1763   COLDVC *=*+1                COLD START IRQ VECTOR
$1764   CPCASE *=*+1                CURRENT PRINTER CASE (HARDCOPY)
$1765   QMODE *=*+1                 HARDCOPY QUOTE MODE
        ;
$1766   RANGNO *=*+1                LINE # RANGE POINTER
$1767   RANGPT *=*+1                LINE # RANGE POINTER
$1768   RANGMX =32
$1768   RANGES *=*+RANGMX           LINE # RANGES
$1788   IERROR *=*+2                VECTOR: PRINT BASIC ERROR MESSAGE
$178A   NUM2 *=*+5
        ;
$178F   ACTLEN *=*+1                VAR FOR GETLIN
$1790   SPELEN *=*+1
$1791   CURPOS *=*+1
        ;
$1792   FSELEN *=*+1                LENGTH OF SEARCH STRING
$1793   FRELEN *=*+1                LENGTH OF REPLACE STRING
$1794   FSKIP *=*+1
$1795   FMODE *=*+1                 BIT7: 1=CHANGE 0=EDIT OR FIND
$1796   RAMFRE *=*+2                POINTER TO FREE IN EXTRA RAM
$1798   RAMTOP *=*+2                TOP OF EXTRA RAM ($C000)
$179A   *=*+10                      UNUSED
        ;
        ;
        MTO =*
        ;
        ;SUBROUTINES TO USE IN ASSEMBLER
        ;CODED SUBROUTINES IN COMAL:
        ;
$17A4   COLD *=*+3                  COLD START OF COMAL
```

```
$17A7   WARM *=*+3          WARM START OF COMAL
$17AA   CALL *=*+3          JSR TO ANOTHER PAGE.
$17AD   GOTO *=*+3          JMP TO ANOTHER PAGE.
$17B0   LOAD *=*+3          LOAD FROM PAGEX
$17B3   STORE *=*+3         STORE TO PAGEX
$17B6   EXEC *=*+3          JSR TO PAGEX
$17B9   SPAGEX *=*+3        SET PAGEX
        ;
$17BC   LDAC1 *=*+3         LOAD AC1
$17BF   LDAC2 *=*+3         LOAD AC2
$17C2   FNDPAR *=*+3        FIND PARAMETER (ASM.CALLS)
$17C5   COPY *=*+9          COPY AREA TOWARDS LOWER ADDRESSES
$17CE   COPYDN *=*+9        COPY AREA TOWARDS HIGHER ADDRESSES
$17D7   FPADD *=*+3         LOAD AC2 AND ADD AC2 TO AC1
$17DA   FPADD2 *=*+9        ADD AC2 TO AC1
$17E3   FPAHF *=*+9         ADD 0.5 TO AC1
$17EC   FPSUB *=*+3         LOAD AC2 AND SUB AC2 FROM AC1
$17EF   FPSUB2 *=*+9        SUB AC2 FROM AC1
$17F8   FPMUL *=*+3         LOAD AC2 AND MUL AC2 BY AC1
$17FB   FPMUL2 *=*+9        MUL AC2 BY AC1
$1804   FPDIV *=*+3         LOAD AC2 AND DIV AC2 BY AC1
$1807   FPDIV2 *=*+9        DIV AC2 BY AC1
$1810   MUL10 *=*+9         MULTIPLY AC1 BY 10.0
$1819   DIV10 *=*+9         DIVIDE AC1 BY 10.0
$1822   STAC1 *=*+3         STORE AC1
$1825   C1T2 *=*+9          COPY AC1 TO AC2
$182E   C2T1 *=*+9          COPY AC2 TO AC1
$1837   FPNEG *=*+9         NEGATE AC1
$1840   FPSGN *=*+9         SIGN OF AC1
$1849   FPSIN *=*+9         SINE OF AC1
$1852   FPCOS *=*+9         COSINE OF AC1
$185B   FPSQR *=*+9         SQUARE ROOT OF AC1
$1864   FPTAN *=*+9         TANGENT OF AC1
$186D   FPPOW *=*+9         RAISE AC2 TO THE POWER OF AC1
$1876   FPATN *=*+9         ARCTANGENT OF AC1
$187F   FPEXP *=*+9         RAISE AC1 TO THE POWER OF E
$1888   FPLOG *=*+9         LOGARITHM BASE E OF AC1
$1891   FPRND *=*+9         COMPUTE PSEUDO-RANDOM NUMBER (RANGE 0 TO 1)
$189A   FPCOM *=*+9         COMPARE NUMBER TO AC1
$18A3   TRUNC *=*+9         CONVERT AC1 INTO INTEGER (-32768 .. 32767)
```

```
        FPINT =TRUNC
$18AC   FPINTG *=*+9        CONVERT AC1 INTO INTEGER (-2^24 .. 2^24-1)
$18B5   FPINTA *=*+9        CONVERT AC1 INTO INTEGER (0 .. 65535)
$18BE   INTFP *=*+9         CONVERT INTEGER INTO FP IN AC1
$18C7   FPASC *=*+9         CONVERT AC1 INTO ASCII EQUIVALENT (STR$)
        ;
$18D0   VAL *=*+6           CONVERT DECIMAL STRING INTO BINARY IN AC1
$18D6   POPA1 *=*+9         POP AC1
$18DF   POPA2 *=*+9         POP AC2
$18E8   PUSHA1 *=*+9        PUSH AC1
$18F1   PUSHRL *=*+9        PUSH REAL NUMBER
$18FA   PSHINT *=*+9        FLOAT & PUSH INTEGER (-32768 .. 32767)
$1903   INTFPA *=*+9        FLOAT & PUSH INTEGER (0 .. 65535)
        ;
$190C   EXCGST *=*+9        ALLOCATE LOCAL STORAGE
$1915   EXCREM *=*+9        RECLAIM LOCAL STORAGE
$191E   RESTOP *=*+9        ALLOCATE GLOBAL STORAGE
$1927   RUNERR *=*+6        GO TO COMAL ERROR HANDLER
        ;
$192D   CRDT *=*+3          READ CHARACTER
$1930   SPACE *=*+2         WRITE SPACE
$1932   CWRT *=*+3          WRITE CHARACTER
$1935   CCHKIN *=*+3        SELECT INPUT FILE
$1938   CCKOUT *=*+3        SELECT OUTPUT FILE
$193B   CCLRCH *=*+3        CLEAR CHANNEL
$193E   CFNAME *=*+7        PARSE & COPY FILE NAME
$1945   COPEN *=*+3         OPEN FILE
$1948   CCLOSE *=*+3        CLOSE FILE
$194B   CRLF *=*+3          OUTPUT CR AND LF
$194E   GETLIN *=*+7        INPUT KEYBOARD LINE
        ;
$1955   RESET *=*+6         RESET PROGRAM POINTERS
$195B   DUMMY *=*+1         EMPTY SUBROUTINE (RTS)
$195C   COMAL *=*+6         GO TO COMAL EDITOR
$1962   EXCUTE *=*+7        EXECUTE CODE IN CDBUF
$1969   JLOAD *=*+7         LOAD COMAL PROGRAM
$1970   ARRLEN *=*+7        COMPUTE NO. OF ELEMENTS IN ARRAY
$1977   FOPEN *=*+7         FAST DISK OPEN
$197E   FCLOSE *=*+7        FAST DISK CLOSE
$1985   CGET *=*+3          GET CHARACTER
```

```
$1988   HARDCO *=*+7                  PRINT HARDCOPY
        ;
        ;-----------------------------------------
        ;
        ;COMAL PROGRAM FOLLOWS HERE:
        ;
        *=$2000
$2000   MBEGIN *=*+4                  START OF MEMORY
$2004   MBEGN1 *=*+1                  START OF NAME TABLE
$2005   MBEGN2 *=*+1                  START OF STACKS
        ;
        ;/////// I/O DEVICE MAP /////////
        ;
        VICCHR =$D000                 VIC CHARACTER ROM
        ;
        VICREG =$D000                 VIC REGISTERS
        ;
        SIDREG =$D400                 SID REGISTERS
        ;
        MMULO =$D500                  MMU PRIMORY REGISTERS
        ;
        VDC =$D600                    8563 REGISTERS
        ;
        VICCOL =$D800                 VIC COLOR NYBBLES
        ;
        CIA1 =$DC00                   6526 #1
        ;
        CIA2 =$DD00                   6526 #2
        ;
        IO1 =$DE00                    EXP. I/O SLOT
        ;
        IO2 =$DF00                    EPX. I/O SLOT RESERVED FOR DMA CTLR
        ;
        MMUHI =$FF00                  MMU SECONDARY REGISTERS
        ;
        ;
        *=VICREG
        ;
$D000   ; 6566 VIDEO INTERFACE CONTROLLER
        ;
```

```
        VIC =*
$D000   SPRPOS *=*+16          SPRITES 0-7 X & Y POS
$D010   SPRXPS *=*+1           SPRITES 0-7 X-POS (MSB OF X-COORD.)
$D011   VCTRL1 *=*+1           VIC CONTROL REGISTER
$D012   RWRAST *=*+1           READ/WRITE RASTER VALUE FOR COMPARE IRQ
$D013   PENX *=*+1             LIGHT-PEN LATCH X-POS
$D014   PENY *=*+1             LIGHT-PEN LATCH Y-POS
$D015   SPRDSP *=*+1           SPRITE DISPLAY ENABLE
$D016   VCTRL2 *=*+1           VIC CONTROL REGISTER
$D017   SPRYEX *=*+1           SPRITES 0-7 EXPAND 2*VERTICAL (Y)
$D018   VCTRL3 *=*+1           VIC MEMORY CONTROL REGISTER
$D019   IRQOCC *=*+1           VIC INTERRUPT FLAG REGISTER
$D01A   IRQMSK *=*+1           IRQ MASK REGISTER
$D01B   SPRBDP *=*+1           SPRITE TO BACKGROUND DISPLAY PRIORITY
$D01C   SPRMCM *=*+1           SPRITES 0-7 MULTI-COLOR MODE SELECT
$D01D   SPRXEX *=*+1           SPRITES 0-7 EXPAND 2*HORIZONTAL (X)
$D01E   SPRSPR *=*+1           SPRITE TO SPRITE COLLISION DETECT
$D01F   SPRBCK *=*+1           SPRITE TO BACKGROUND COLLISION DETECT
$D020   BORCOL *=*+1           BORDER COLOR
$D021   BCKCOL *=*+4           BACKGROUND COLOR 0-3
$D025   SPRMCL *=*+2           SPRITE MULTI-COLOR REGISTER 0-1
$D027   SPRCOL *=*+8           SPRITE 0-7 COLOR
        ;
        *=SIDREG
        ;
$D400   ; 6581 SOUND INTERFACE DEVICE
        ;
        SID =*                 VOICE 1:
$D400   V1FREQ *=*+2           FREQUENCY CONTROL
$D402   V1PWW *=*+2            PULSE WAVEFORM WIDTH
$D404   V1CTRL *=*+1           CONTROL REGISTER
$D405   V1ENVL *=*+2           ENVELOPE GENERATOR (ADSR)
        ;VOICE 2:
$D407   V2FREQ *=*+2           FREQUENCY CONTROL
$D409   V2PWW *=*+2            PULSE WAVEFORM WIDTH
$D40B   V2CTRL *=*+1           CONTROL REGISTER
$D40C   V2ENVL *=*+2           ENVELOPE GENERATOR (ADSR)
        ;VOICE 3:
$D40E   V3FREQ *=*+2           FREQUENCY CONTROL
$D410   V3PWW *=*+2            PULSE WAVEFORM WIDTH
```

```
$D412   V3CTRL *=*+1                   CONTROL REGISTER
$D413   V3ENVL *=*+2                   ENVELOPE GENERATOR (ADSR)
        ;FILTERS:
$D415   FCUTOF *=*+2                   FILTER CUTOFF FREQUENCY
$D417   FRESON *=*+1                   FILTER RESONANCE/VOICE INPUT CONTROL
$D418   FMOVOL *=*+1                   SELECT FILTER MODE AND VOLUME
$D419   PADDL1 *=*+1                   A/D-CONVERTER: GAME PADDLE 1
$D41A   PADDL2 *=*+1                   A/D-CONVERTER: GAME PADDLE 2
$D41B   OSC *=*+1                      OSCILLATOR 3 RANDOM NUMBER GENERATOR
$D41C   ENV *=*+1                      ENVELOPE GENERATOR 3 OUTPUT
        ;
        ;VIC COLOR RAM
        ;
        *=VICCOL
$D800   COLRAM *=*+SCSIZE+24           COLOR RAM (NYBBLES)
        ;
        *=CIA1
        ;
$DC00   ;6526 COMPLEX INTERFACE ADAPTER #1
        ;
        CHTIM =*
        COLM =*                        KEYBOARD MATRIX
$DC00   D1PRA *=*+1
        ROWS =*                        KEYBOARD MATRIX
$DC01   D1PRB *=*+1
$DC02   D1DDRA *=*+1
$DC03   D1DDRB *=*+1
$DC04   D1T1L *=*+1
$DC05   D1T1H *=*+1
$DC06   D1T2L *=*+1
$DC07   D1T2H *=*+1
$DC08   D1TOD1 *=*+1
$DC09   D1TODS *=*+1
$DC0A   D1TODM *=*+1
$DC0B   D1TODH *=*+1
$DC0C   D1SDR *=*+1
$DC0D   D1ICR *=*+1
$DC0E   D1CRA *=*+1
$DC0F   D1CRB *=*+1
        ;
```

```
          *=CIA2
          ;
$DD00     ;6526 COMPLEX INTERFACE ADAPTER #2
          ;
$DD00     D2PRA *=*+1
$DD01     D2PRB *=*+1
$DD02     D2DDRA *=*+1
$DD03     D2DDRB *=*+1
$DD04     D2T1L *=*+1
$DD05     D2T1H *=*+1
$DD06     D2T2L *=*+1
$DD07     D2T2H *=*+1
$DD08     D2TOD1 *=*+1
$DD09     D2TODS *=*+1
$DD0A     D2TODM *=*+1
$DD0B     D2TODH *=*+1
$DD0C     D2SDR *=*+1
$DD0D     S2ICR *=*+1
$DD0E     D2CRA *=*+1
$DD0F     D2CRB *=*+1
          ;
          *=IO1
          ;
$DE00     OVRLAY *=*+256              OVERLAY CONTROL PORT
          ;
$D500     ;C/128 MEMORY MANAGEMENT UNIT
          ;
          *=MMULO
$D500     MMUCRL *=*+1                CONFIGURATION REGISTER (PRIM)
$D501     PCRA *=*+1                  PRE. CONFIG. REG. A
$D502     PCRB *=*+1                  PRE. CONFIG. REG. B
$D503     PCRC *=*+1                  PRE. CONFIG. REG. C
$D504     PCRD *=*+1                  PRE. CONFIG. REG. D
$D505     MMUMCR *=*+1                MODE CONFIG. REG.
$D506     MMURCR *=*+1                RAM CONFIG. REG.
$D507     MMUP0L *=*+1                PAGE 0 PTR LOW
$D508     MMUP0H *=*+1                PAGE 0 PTR HIGH
$D509     MMUP1L *=*+1                PAGE 1 PTR LOW
$D50A     MMUP1H *=*+1                PAGE 1 PTR HIGH
$D50B     MMUVER *=*+1                MMU VERSION NUMBER
```

```
          ;
          *=MMUHI
$FF00     MMUCR *=*+1                 CONFIG. REG. SECONDARY
$FF01     LCRA *=*+1                  LOAD CONFIG. REG. A
$FF02     LCRB *=*+1                  LOAD CONFIG. REG. B
$FF03     LCRC *=*+1                  LOAD CONFIG. REG. C
$FF04     LCRD *=*+1                  LOAD CONFIG. REG. D
          ;
          ;
$D600     ; C/128 80-COLUMN VIDEO CONTROLER
          ;
          *=VDC
$D600     VDCADR *=*+1                8563 ADDR. REG
$D601     VDCDAT *=*+1                8563 DATA REG.
          ;
          VDCSCN =$0000               8563 80-COLUMN SCREEN (2KB)
          VDCCOL =$0800               8563 ATTRIBUTE AREA (2KB)
          VDCCHR =$2000               8563 CHARACTER RAM (4KB)
          ;
          SKIP =$2C                   OPCODE FOR 'BIT <ABS>' (SKIP 2 BYTES)
          ;
          ;
          ;*********************************************************
          ;
          ;
          ;FOR USE IN ASSEMBLER CODED SUBROUTINES IN COMAL:
          ;
          ;LABELS FOR PROC/FUNC DEFINITIONS
          ;
          ROMMED =%10000000
          ;
          DEFPAG =2                   RAM0+I/O
          ;
          PROC =112
          ENDPRC =126
          ;
          FUNC =227
          ENDFNC =126
          ;
          PARAM =114
```

```
VALUE =PARAM+0
REF =PARAM+3
ARRAY =6
;
REAL =0
INT =1
STR =2
;
;
;SIGNAL TYPES:
;
POWER1 =0          POWER UP SEI (ONLY FOR ROMMED LIBRARIES)
POWER2 =1          POWER UP CLI (ONLY FOR ROMMED LIBRARIES)
LINK =2            AFTER LINK/LOAD-COMMAND (ONLY LIBRARY JUST LINKED)
DSCRD =3           BEFORE DISCARD-COMMAND
NEW =4             AFTER NEW-COMMAND OR BAD LOAD/RUN/CHAIN
CLRTAB =5          AFTER NAME TABLE IS CLEARED
RUN =6             BEFORE RUN/CHAIN
WARM1 =7           WARM START SEI
CON =8             BEFORE CONTINUATION (CON COMMAND)
ERROR =9           AFTER ERROR MESSAGE PRINTED
STOP1 =10          AFTER STOP/END
BASIC =11          BEFORE LEAVING COMAL
;
;
.END
=
```