RCBD UDOS

Version 2

Ralf-Peter Nerlich
rio@early8bitz.de

7. Februar 2026

Zusammenfassung

Handbuch RCBD

UDOS auf RomWBW unterstützter Hardware

Dieses Dokument wurde mit LaTeX 2_ε formatiert.

Inhaltsverzeichnis

1 Sicherheitshinweise

Beachten Sie beim Umgang mit Halbleitern und elektronischen Baugruppen die allgemeinen Richtlinien zum Umgang mit elektrostatisch empfindlichen Bauelementen (ESD sensitive Devices).

RC2014, RCBus und ähnliche Systeme, die als Zielplattform für diese Software dienen, haben keine Hotplug-Fähigkeiten. Der Wechsel von Baugruppen oder Datenträgern muss daher unbedingt im spannungslosen Zustand erfolgen.

2 Änderungen, Changelog

Initiale Version V2 (veröffentlicht):
RCBD-ROM 250528 V2, Softwarepaket 2025-07-17, Handbuch 2025-08-11

Änderungen:

- 2025-08-03: Neues Dienstprogramm **'romwrite'** (siehe 11.6). Rückwärtskompatibel zum *RCBD-ROM* 250528 V2. **'romwrite'** ist bereits im Handbuch 2025-08-11 beschrieben aber noch nicht im Softwarepaket 2025-07-17 enthalten.
- 2025-08-16: *RCBD-ROM* 250816 V2 (intern)
- 2025-08-16: Dienstprogramm **'md'**:
  Option `[<device>]*` zum Anzeigen und Ändern des Default-Harddisk-Device (siehe 9.1.4). Erfordert *RCBD-ROM* 250816 V2 oder neuer.
- 2025-08-27: *RCBD-ROM* 250827 V2 (intern)
  Fix für Slicehandling bei *RomWBW*-Partitionen ≥ 2 GiB
- 2025-09-08: Dienstprogramm **'sysinfo'**:
  Programm zum Anzeigen verschiedener Systeminformationen (siehe 11.13).
- 2025-10-08: *RCBD-ROM* 251008 V2 (intern)
  Fix für manuelle Programmunterbrechung (Break) mit NMI-Taste.
- 2025-10-09: *RCBD-ROM* 251009 V2 (intern)
  Plattform MSX (PLT_MSX, ID=25) zugefügt. Ab *RomWBW*-Version 3.6.0-dev.30 enthalten.
- 2025-10-18: *RCBD-ROM* 251018 V2 (intern)
  Codeoptimierungen; Zählung C5- und C6-Fehler im `$FLOPPY`-Treiber.
- 2025-11-11: *RCBD-ROM* 251111 V2 (intern)
  Verbesserung Interruptstabilität

- 2025-11-25: *RCBD-ROM* 251125 V2 (intern)
  Anzahl verwendbarer *HBIOS*-Devices von 8 auf 10 erhöht.
- 2025-11-25: Dienstprogramm **'md'**:
  Anpassung an Erhöhung der verwendbaren HBIOS-Devices ab *RCBD-ROM* 251125 V2. Rückwärtskompatibel bis *RCBD-ROM* 250816 V2.
- 2025-11-25: Dienstprogramm **'sysinfo'**:
  Anpassung an Erhöhung der verwendbaren HBIOS-Devices ab *RCBD-ROM* 251125 V2. Rückwärtskompatibel bis *RCBD-ROM* 250816 V2.
- 2025-12-04: Dienstprogramm **'copy.disk'**:
  Option `q=n` zur Unterdrückung der Sicherheitsabfrage, Option `v` für Verbose Modus.
- 2025-12-06: Dienstprogramm **'format'**:
  Überarbeitung Kommandozeilenauswertung
- 2026-01-20: *RCBD-ROM* 260120 V2 (veröffentlicht)
  Fixes in *RCBD-ROM*-Monitor/Debugger Breakpointlogik.
- 2026-01-27: Dienstprogramm **'romwrite'**:
  Neue Option zur Manipulation des Labels im Boot Info Block. Mehr Diagnoseausgaben (verbose).

Update Version V2 (veröffentlicht):
RCBD-ROM 260120 V2, Softwarepaket 2026-01-27, Handbuch 2026-02-07

3 Einleitung

RCBD ist eine Portierung des 8-Bit-Betriebssystems *UDOS* 3.1 des EAW-Computers P8000[1] auf moderne Z80- und Z180-basierte modulare oder Singleboardcomputer. Die Implementierung des Betriebssystems setzt auf der *HBIOS*-Schnittstelle von *RomWBW*[2] auf, das für viele Retrocomputingsysteme verfügbar ist.

Der Name *RCBD* leitet sich von **RCB**us-**D**USSEL ab. DUSSEL war der Name meines Selbstbaucomputers aus den 80er Jahren[3], der bzgl. *UDOS* ebenfalls annäherend P8000-kompatibel war.

Prinzipiell sollte die *RCBD*-Implementierung auf jeder Z80- oder Z180-basierenden Hardware lauffähig sein, die von *RomWBW* unterstützt wird und die die unter Punkt 5.1 gelisteten Minmalvoraussetzungen erfüllt.

Um *RCBD* auf einen *RomWBW*-Rechner zu bringen, sind einige Grundkenntnisse im Umgang mit der *RomWBW*-Architektur notwendig. Das betrifft z. B. das Generieren einer kundenspezifischen *RomWBW* ROM-Version, *RomWBW*-Firmware flashen und updaten, Präparierung von Datenträgern zur Nutzung unter *RomWBW*. Daher sei hier auf die ausführliche Dokumentation zu *RomWBW* verwiesen, die im Git-Repository[2] des Projekts als PDF zu finden ist.

- *RomWBW* Introduction
- *RomWBW* System Guide
- *RomWBW* User Guide
- Hard Disk Anatomy

Hinweis: Auch wenn die Systemphilosophie bzgl. Datenträgerformaten, Dateisystem usw. vom P8000 übernommen wurde, sind die Systemdisketten von *RCBD* und P8000 nicht wechselseitig bootfähig. Der größte Teil der allgemeinen Dienst- und Anwendungsprogramme ist jedoch auf beiden Plattformen lauffähig.

[1] https://www.robotrontechnik.de/index.htm?/html/computer/p8000.htm
[2] https://github.com/wwarthen/RomWBW/
[3] https://rio.early8bitz.de/__fps2/fps2-computer.htm

4 Begriffserklärung

`RomWBW`

Die *RomWBW*-Software[2] bietet eine vollständige Implementierung von CP/M (und ähnlichen) Betriebssystemen und eigenständigen Anwendungen für moderne Z80/Z180/Z280 Retrocomputinghardware.

`HBIOS`

RomWBW isoliert konzeptgemäß alle hardwarespezifischen Funktionen im ROM-Chip selbst. Das ROM bietet eine Hardwareabstraktionsschicht, das *HBIOS*, so dass alle Betriebssysteme und eigenständigen Anwendungen auf einem Diskdevice auf jedem *RomWBW*-basierten System laufen.

`HBIOS-Proxy (HBX)`

Während das *HBIOS* selbst versteckt in einer separaten Speicherbank ausgeführt wird, bildet der permanent verfügbare *HBIOS*-Proxy (kurz HBX) die Aufrufschnittstelle für alle *HBIOS*-Funktionen.

`RomWBW-Monitor`

Eigenständiges Programm, das direkt vom *RomWBW*-Bootprompt gestartet werden kann. Es ermöglicht das Laden von Programmen (Intel-HEX oder XMODEM) in den Speicher von *RomWBW* und das Manipulieren des Speicherinhaltes.

`RomWBW-Loader`

RomWBW ermöglicht das Starten von binär vorliegenden Programmimages direkt aus einem Slice eines unterstützten Datenträgers. Dazu muss der Binärcode des zu startenden Programmes zusammen mit einem Präfix (Boot Info Block), das die Ladeinformationen enthält, im betreffenden Slice abgelegt werden.

`Slice`

Zentrales Element zur Verwaltung *RomWBW* unterstützter Datenträger in kleineren Einheiten zur Präsentation als Laufwerk an die laufenden Betriebssysteme oder eigenständigen Programme. In Punkt 6.4 weiter erläutert.

`RIO`

RIO ist ein diskettenorientiertes Betriebssystem, das Zilog Mitte der 70er Jahre zusammen mit seinen Mikrorechner-Entwicklungssystemen (ZDS, MCZ, PDS) auf den Markt brachte. Der Fokus dieser Systeme lag in der Bereitstellung von Hardware- und Software-Entwicklungswerkzeugen für die eigenen Prozessorfamilien Z80, Z8 und Z8000. Für den Einsatz als Personalcomputer im Büroumfeld waren diese Geräte mangels officespezifischer Anwendungsprogramme weniger geeignet.

`UDOS`

UDOS ist die Portierung von *RIO* auf in der damaligen DDR hergestellte Computerhardware, die mit der Z80 kompatiblen Prozessorfamilie U880 ausgestattet war[4]. Die ersten *UDOS*-Versionen arbeiteten mit dem übernommenen *RIO* Dateisystemtreiber `$ZDOS`, der ein propritäres Sektorformat zur Verlinkung von physischen Sektoren zu logischen Dateien verwendete. Spätere *UDOS*-Versionen benutzten ein standardisiertes Sektorformat sowie Zeigerblöcke, um die zu einer Datei gehörenden physischen Sektoren zu verwalten. Zu diesen Versionen gehört *UDOS* 5 auf dem A5120 oder PC1715W und *UDOS* 3.1 auf dem EAW P8000.

`RCBD`

RCBD ist ein Hobbyprojekt, dass es zum Ziel hat, *UDOS* auf heute im Retrocomputingumfeld verwendete Hardware zu portieren. Ausgangspunkt ist dabei das Betriebssystem *UDOS* 3.1 des EAW P8000.

`RCBD-ROM`

RCBD-ROM ist die residente Komponente des *RCBD*. Sie entspricht funktionell dem 3 KiB ROM der Zilog-Systeme bzw. den 3 KiB residenter Software, die bei den DDR-*UDOS*-Implementierungen von speziell reservierten Sektoren der Systemdiskette oder einem versteckten ROM in den unteren Teil des Hauptspeichers geladen wurde.

RCBD-ROM enthält die residententen Treiber `$FLOPPY` und `$PCON`.

`RCBD-UDOS`

RCBD-UDOS ist die ladbare Komponente des *RCBD*. Sie entspricht im wesentlichen dem vom P8000 bekannten *UDOS* 3.1. Einige Systemkomponenten und Dienstprogramme mussten für die Portierung modifiziert oder neu geschrieben werden. Diese Komponenten werden in den Kapiteln 10 und 11 beschrieben.

Da es aus Sicht der Betriebssystemphilosophie keine gravierenden Unterschiede zwischen *RIO*, *UDOS* und *RCBD-UDOS* gibt, soll im weiteren nur noch der Begriff *UDOS* für das ladbare bzw. geladene Betriebssystem mit seinen Dienst- und Anwendungsprogrammen benutzt werden.

`$FLOPPY`

Low-Level- (Sektor-Level-) Treiber zur physischen Bedienung der Diskettenlaufwerke. Im *RCBD* übernimmt `$FLOPPY` auch das Bedienen der RAM-Disk. Im *RCBD* spricht `$FLOPPY` die Hardware der Massenspeicher

[4] https://www.robotrontechnik.de/html/software/udos.htm

nicht direkt an, sondern wandelt die Requests in entsprechende *HBIOS*-Aufrufe zum Zugriff auf die tatsächlichen Datenträger um. `$FLOPPY` ist Bestandteil des *RCBD-ROM* und somit unmittelbar nach dem Starten des *RCBD* aktiv.

`$NDOS`

Treiber zur logischen Verwaltung des *UDOS* Dateisystems. `$NDOS` ist hardwareneutral, es benutzt Aufrufe an `$FLOPPY` zum Zugriff auf die einzelnen Sektoren der Disketten. `$NDOS` ist eine separate Datei (**'NDOS'**) die während des Bootvorgangs von *UDOS* in den Speicher geladen wird.

`$PCON`

Treiber zur physischen Bedienung der seriellen Konsolenschnittstelle. Im *RCBD* spricht `$PCON` die Hardware der seriellen Schnittstelle nicht direkt an, sondern wandelt die Requests in entsprechende *HBIOS*-Aufrufe zum Zugriff auf die tasächlichen seriellen Bausteine um. `$PCON` ist Bestandteil des *RCBD-ROM* und somit unmittelbar nach dem Starten des *RCBD* aktiv.

`$CON`

Erweiterter Treiber für die serielle Konsolenschnittstelle. `$CON` erweitert den Funktionsumfang des `$PCON` Treibers. `$CON` ist hardwareneutral, es benutzt Routinen aus `$PCON` beim Zugriff auf die serielle Hardware. `$CON` ist integrierter Bestandteil der Betriebssystemexekutive (Datei **'OS'**) und somit erst nach Laden von *UDOS* aktiv.

`$PRTC`

Treiber zum Zugriff auf die Datums- und Zeitinformationen der Echtzeituhr. Im *RCBD* spricht `$PRTC` die Hardware der Echtzeituhr nicht direkt an, sondern wandelt die Requests in entsprechende *HBIOS*-Aufrufe zum Zugriff auf die tatsächliche RTC-Hardware um. `$PRTC` ist Bestandteil des *RCBD-ROM* und somit unmittelbar nach dem Starten des *RCBD* aktiv.

5 Systemvoraussetzungen

5.1 8-Bbit-Rechner

Mindestens notwendig sind:

- Z80-CPU oder Z180-CPU im Interruptmode 2
- *RomWBW* unterstützer Paged ROM
- *RomWBW* unterstützer Paged RAM; Empfohlen 512 KiB Paged RAM (sechzehn 32 KiB Seiten) für Nutzung einer RAM-Disk (Standard RAM-Bank-Layout). Auch 128 KiB Paged RAM (vier 32 KiB Seiten) möglich, dann ohne RAM-Disk (Tiny RAM-Bank-Layout).
- *RomWBW* unterstützter Memorymanager (MMU)
- Mind. ein serieller Kanal (Z80-SIO, ACIA, UART, interner ASCI bei Z180) für Systemkonsole
- Mind. ein interruptfähiger Timer (CTC bei Z80, interner Timer bei Z180)
- Mind. ein Massenspeicher incl. Controller, der von *RomWBW* als Harddisk reportet wird (CF-Karte, SD-Karte, pATA (Parallelport IDE), USB-Stick oder -Festplatte)
- *RomWBW*-Firmware Version 3.4.0 oder höher

Weiterhin kann *RCBD* bisher folgende optionale Hardware nutzen:

- *RomWBW* unterstütze Echtzeituhr (RTC)
- Prellfreie Taste (Lowaktiv) am /NMI-Anschluss (Break-Taste für manuelle Programmunterbrechung, siehe Anlage L.3)

Noch nicht implementiert ist die Einbindung folgender Hardware in *RCBD*

- Zweiter serieller Kanal zum Anschluss eines Druckers

Alle weitere Hardware, die von *RomWBW* unterstützt wird, ist unter *RCBD* nicht zugänglich.

5.2 Serielles Terminal

UDOS und so auch *RCBD* arbeitet konsoleorientiert. Zur Bedienung ist ein serielles Terminal notwendig, das bzgl. Baudrate auf die genutzte Schnittstelle einstellbar sein muss. Das kann ein Hardwareterminal sein, oder ein externer Computer mit Terminalemulationsprogramm. Die Einstellungen für die Übertragungsparameter stehen im Anlage A.

Das eigentliche Betriebssystem verlangt keinen speziellen Terminaltyp, da alle Dienstprogramme rein zeilenorientiert arbeiten. Lediglich der auf der Systemdiskette enthaltene (selbst geschriebene) Bildschirmeditor **'vi'** (11.15) erfordert ein ANSI kompatibles Terminal (z. B. VT100).

In *RCBD-UDOS* existiert eine Implementierung des XMODEM Programms über die serielle Konsolenschnittstelle, um Dateien zwischen dem *RCBD*-System und einem externen Computer zu übertragen. Zur Nutzung muss das verwendete Terminalemulationsprogramm das XMODEM Protokoll implementiert haben, was z. B. bei TeraTerm[5] der Fall ist.

[5] https://teratermproject.github.io/index-en.html

6 UDOS auf RomWBW/HBIOS

6.1 CPU

6.1.1 Unterstützte CPUs

Obwohl *RomWBW* neben den Z80- und den Z180-basierten Rechnern auch einige Retrocomputingsysteme mit eZ80 und Z280 unterstützt, beschränkt sich die *RCBD*-Implementierung auf Rechner mit den beiden CPUs Z80 und Z180.

Die Technologie (NMOS/CMOS) und die Taktfrequenzen der eingesetzten CPUs spielen für *RCBD* nur eine untergeordnete Rolle. Generell ist natürlich schneller und stromärmer immer die bessere Wahl.

6.1.2 UDOS auf Z180 - geht das?

Ja. Die Z180-CPU ist weitestgehend befehlskompatibel zur Z80-CPU. Die zusätzlichen Befehle der Z180-CPU spielen nur bei der Systeminitialisierung und einigen hardwarenahen Passagen eine Rolle, was vollständig durch das *RCBD-ROM* und das *HBIOS* verdeckt ist. Das *UDOS*-Betriebssystem sowie die Anwendungs- und Dienstprogramme benötigen diese Befehle nicht. Somit ist die *RCBD*-Systemdiskette zwischen Z80- und Z180-basierten Rechnern uneingeschränkt austauschbar.

Wie sieht es mit den undokumentierten Befehlen der Z80-CPU aus? Diese sind nicht in der Z180-CPU implementiert. Alle nicht dokumentierten Opcodes führen beim Z180 zu einer Ausnahmebehandlung (Trap), die im *RCBD-ROM* abgefangen und gemeldet wird. Zilog hat in seinem *RIO*-Betriebssystem nie die undokumentierten Befehle verwendet, und auch bei der Portierung des P8000-*UDOS* zum *RCBD* ist mein Z180-System noch nie in einen Trap gelaufen.

Durch *UDOS*-Anwender selbst geschriebene Software könnte natürlich undokumentierte Z80 Befehle enthalten. Aber *UDOS* selbst mit seinen eigenen Programmen scheint diesbezüglich 'sauber' zu sein.

6.2 Hauptspeicher

UDOS kennt nur 64 KiB linearen Hauptspeicher. In der *RomWBW* Umgebung wird dies durch die User-Bank in den unteren 32 KiB des Hauptspeichers (`0x0000..0x7FFF`) bereitgestellt. Die oberen 32 KiB (`0x8000..0xFFFF`) werden durch die Common-Bank gebildet. Diese Speicherpräsentation darf sich während der Laufzeit von *UDOS*-Code nicht ändern.

Während die Common-Bank permanent im oberen Adressbereich eingeblendet ist, kann die User-Bank temporär durch eine andere Bank ersetzt wer-

den (Bank Switching). Das passiert z. B. immer, wenn die Anwendungssoftware (in unserem Fall ein für *RCBD* angepasster *UDOS*-Treiber) mit der Hardware kommunizieren will. Dann wird über einen Aufruf einer *HBIOS*-Funktion der entsprechende *RomWBW*-Treiber benutzt. Diese Treiber liegen in der *HBIOS*-Bank, die zu diesem Zweck in die unteren 32 KiB eingeblendet wird. D. h. der untere Teil des *UDOS*-Speichers ist während dessen 'weg'.

Damit der Aufruf und vor allem die Rückkehr in das aufrufende *UDOS*-Programm funktioniert, existiert eine Zwischenstation in der permanent verfügbaren Common-Bank, der sogenannte *HBIOS*-Proxy (HBX). Dieser liegt am oberen Ende des 64 KiB Adressraumes und belegt 512 B. Bei einem Aufruf einer *HBIOS*-Funktion passiert etwa folgendes:

- Der aufrufende *UDOS*-Treiber (egal wo er im Hauptspeicher liegt) ruft eine passende *HBIOS*-Funktion mit den notwendigen Parametern über den HBIOS-Proxy auf.
- Der HBX schaltet die unteren 32 KiB des Hauptspeichers von der User-Bank auf die *HBIOS*-Bank um.
- Der HBX ruft die dem Funktionsaufruf entsprechende Routine in der *HBIOS*-Bank auf.
- Am Ende führt die *HBIOS*-Routine ein Return zur rufenden Instanz im HBX durch.
- Der HBX schaltet die unteren 32 KiB des Hauptspeichers zurück von der *HBIOS*-Bank auf die User-Bank, damit sind die 64 KiB von *UDOS* wieder komplett.
- Der HBX führt ein Return zum rufenden Programm durch, welches sich jetzt wieder an der ursprünglichen Stelle im Hauptspeicher befindet.

Da *UDOS*-Programme die Hardware nicht direkt ansprechen, sondern über entsprechende *UDOS*-Treiber, ist das Verhalten für das *UDOS*-Betriebssystem (OS) und reine *UDOS*-Dienstprogramme transparent. Im wesentlichen betrifft dies die beiden Treiber `$FLOPPY` für physische Diskettenzugriffe und `$PCON` für die Ein- und Ausgabe über die serielle Konsolenschnittstelle. Beide Treiber sind im residenten Teil von *UDOS* dem *RCBD-ROM* enthalten und verdecken die *RomWBW*-Anpassung vor dem eigentlichen *UDOS*-Betriebssystem und seinen Dienst- und Anwendungsprogrammen.

Einen Überblick über die Speicheraufteilung im *RCBD* gibt Anlage B.

6.3 Stack

Die Konsequenz aus dem Paging der unteren 32 KiB zwischen *UDOS*-Speicher und *HBIOS* ist, dass der Stackpointer der CPU stets auf einen permanent verfügbaren Speicher, also auf eine Adresse in den oberen 32 KiB (Common-Bank ab `0x8000`) zeigen muss. Für *RCBD-ROM*, **OS** und **NDOS** ist das sicher gestellt. Ladbare Dienst- und und Anwendungsprogramme, die sich gemäß *UDOS*-Konzept den Stack vom **OS** dynamisch zuteilen lassen, sind unverändert lauffähig, da die Zuteilungsalgorithmen im **OS** (Punkt 10.2) angepasst wurden. Ladbare Programme, die einen hart kodierten Stack in den unteren 32 KiB verwenden, müssen manuell gepatched werden. Der Stack dieser Programme muss in den extra dafür reservierten 256 B Bereich `0xFB00..0xFBFF` verlegt werden (siehe Anlage B).

6.4 Massenspeicher

UDOS arbeitet ausschließlich mit Disketten und kann 8 Diskettenlaufwerke (Laufwerke 0...7) adressieren. Der als Vorbild dienende P8000 hat einen 8272-basierten Diskettenkontroller, welcher nur vier Laufwerke (0...3) bedienen kann. Im *RCBD* wurde der `$FLOPPY`-Treiber um die Möglichkeit erweitert, jedes der vier Laufwerke entweder als klassisches Diskettenlaufwerk (Floppy-Disk) oder als RAM-Disk zu konfigurieren. In der bekannten *RomWBW*-kompatiblen Hardware kann maximal eine RAM-Disk existieren, sofern die Hardware mit einem 512 KiB RAM-Baustein ausgestattet ist (bei kleinerem RAM sind nicht genug Bänke für eine RAM-Disk verfügbar).

RomWBW organisiert den Massenspeicher in sogenannten Slices zu je 8 MiB. Das hat seinen Ursprung in den in *RomWBW* serienmäßig mitgelieferten CP/M-Betriebssystemen, die bis zu 8 MiB pro Laufwerk nutzen können. Auf diese Organisation wurde bei *RCBD* aufgebaut, obwohl das Image einer *UDOS*-Diskette nur 640 KiB groß ist. Bei einer Maxmimalanzahl von 256 Slices (davon 255 nutzbar von *RCBD*) bei einem Datenträger ≥ 2 GiB ist Platz für wahrscheinlich alle noch existierenden *UDOS*-Disketten. Der Platzverschleiß wurde zu Gunsten einer leichteren Umrechnung einer *UDOS*-Diskadresse (Track/Sektor) in die LBA-Adresse eines bestimmten Slices auf dem physischen Speichermedium in Kauf genommen. Außerdem ermöglicht erst die simple Zuordnung 'Ein Slice = Eine *UDOS*-Diskette' ein vernünftiges Sichern einzelner Diskettenimages auf einen externen Rechner oder das Einspielen eines weiteren Diskettenimages in einen freien Slice eines *RomWBW*-Datenträgers.

RCBD akzeptiert nur Datenträger die

- von RomWBW als Harddisk reportet werden (CF, SD, pATA, USB) und im 'Modern Layout' (hd1k) vorbereitet sind
- von RomWBW als RAM-Disk mit mindestens 256 KiB reportet werden

- und in beiden Fällen als LBA adressierbar reportet werden

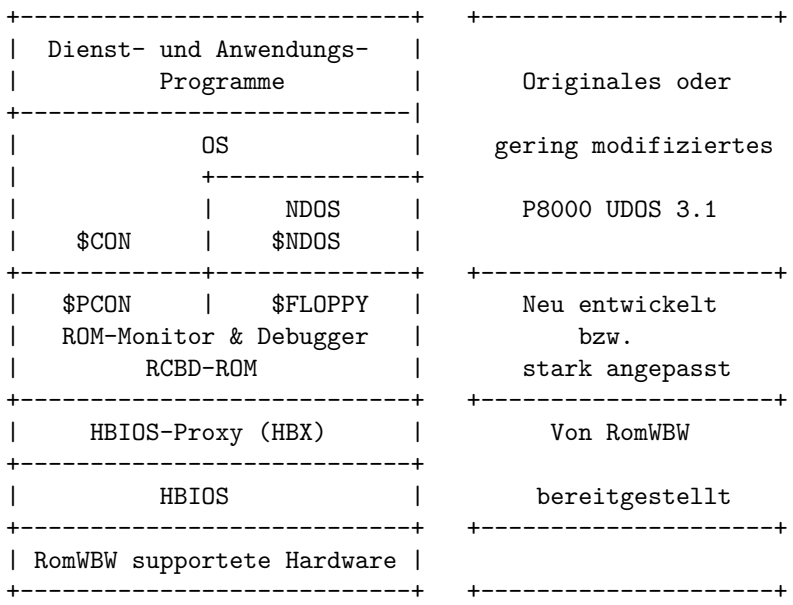
Native Floppy-Disks und Harddisks, die im älteren 'Classic Layout' (hd512) präpariert sind, werden nicht unterstützt.

Grundlagen zu Datenträgern findet man im Kapitel 'Disk Layout' im '*RomWBW* System Guide' sowie für die *RomWBW* konforme Vorbereitung neuer Datenträger im Kapitel 'Disk Preparation' im '*RomWBW* User Guide'. Anlage C listet typische Datenträgergrößen auf. Die Vorbereitung eines Beispieldatenträgers ist in diesem Handbuch in Anlage N beschrieben.

Als Diskettenformat wird im *RCBD* ausschließlich der beim P8000 als Typ 5 bezeichnete Diskettentyp (80 Spuren, 2 Köpfe (Double Side), 16 Sektoren pro Kopf und Seite, 256 B pro Sektor) unterstützt. Das entspricht dem in der DDR als K5601 bzw. MFS 1.6 bekannten Laufwerk. Die Kapazität ist 640 KiB (2560 Sektoren @ 256 B) pro Diskette.

6.5 Hierarchie der RCBD Komponenten

Die einzelnen Ebenen in der Struktur von *RCBD* sind in der folgenden Grafik dargestellt.



7 Systemstart

Der Systemstart des *RCBD* besteht aus drei Teilschritten.

- Starten der *RomWBW*-Firmware nach Power On oder Reset
- Laden des residenten Teils von *RCBD* (*RCBD-ROM*) in den RAM und Starten des *RCBD-ROM*-Monitors
- Booten des *UDOS*-Betriebssystems von Diskette

7.1 Start RomWBW

RomWBW startet nach Power On oder Reset (Taste), kopiert die wesentlichen Teile der Firmware (*HBIOS*, *HBIOS*-Proxy) in den RAM und führt eine Hardwareerkennung und -initialisierung durch. Die konfigurierte u/o gefundene Konfiguration wird detailiert ausgegeben. Ebenso werden alle erkannten Massenspeicher aufgelistet.

Der folgende Screenshot zeigt die Startmeldungen vom Start eines SC126[6] Z180-Single-Board-Computers. Es finden sich zwei serielle Schnitstellen (ASCI0: und ASCI1:) sowie acht Disk Devices. Neben der RAM-Disk (MD0:) und der ROM-Disk (MD1:) in den Speicherchips des SBC steckt ein Compact Flash Modul SC729[7] als Erweiterungskarte auf dem SBC, bestückt mit einer 128 MB CF-Karte (IDE0:) und einer 256 MB CF-Karte (IDE1:). Außerdem steckt noch ein SD-Kartenadapter auf dem SBC, der mit einer 4 GB SD-Karte bestückt ist (SD0:). Dazu kommt ein Parallelport-IDE-Adapter[8] mit einer 128 MB Festplatte (PPIDE0:) Ein CH376-basierter USB-Controller[9] mit einem 500 MB USB-Stick (CHUSB0:) vervollständigt die Massenspeicherausstattung.

Die Konfiguration des *RomWBW*-ROMs steht in Anlage E.

[6] https://smallcomputercentral.com/sc126-z180-motherboard-rc2014/
[7] https://smallcomputercentral.com/sc729-rcbus-compact-flash-module/
[8] https://rc2014.co.uk/modules/ide-hard-drive-module/
[9] https://rc2014.co.uk/modules/ch375-usb-storage/

```
------------------------------------------------------------------------
RomWBW HBIOS v3.5.1, 2025-07-10

Small Computer SC126 [SCZ180_sc126_std] Z8S180-N @ 18.432MHz IO=0xC0
0 MEM W/S, 2 I/O W/S, INT MODE 2, Z180 MMU
512KB ROM, 512KB RAM, HEAP=0x2487
ROM VERIFY: 00 00 00 00 PASS

ASCI0: IO=0xC0 ASCI W/BRG MODE=115200,8,N,1
ASCI1: IO=0xC1 ASCI W/BRG MODE=115200,8,N,1
DSRTC: MODE=STD IO=0x0C Sun 2025-12-07 22:39:10 CHARGE=OFF
MD: UNITS=2 ROMDISK=384KB RAMDISK=256KB
FD: MODE=RCWDC IO=0x50 NOT PRESENT
IDE: IO=0x10 MODE=RC
IDE0: ATA 8-BIT LBA BLOCKS=0x0003D400 SIZE=122MB
IDE1: ATA 8-BIT LBA BLOCKS=0x0007A2B0 SIZE=244MB
PPIDE: IO=0x20
PPIDE0: ATA LBA BLOCKS=0x0EE7C2B0 SIZE=122104MB
PPIDE1: NO MEDIA
SD: MODE=SC OPR=0x0C CNTR=0xCA TRDR=0xCB DEVICES=1
SD0: SDHC NAME=SU04G BLOCKS=0x00762C00 SIZE=3781MB
CH0: IO=0x3E TYPE=CH376 VER=0x01
CHUSB0: BLOCKS=0x000F9FFF SIZE=499MB
CH1: IO=0x3C NOT PRESENT
FP: IO=0x00 NOT PRESENT

Unit        Device      Type              Capacity/Mode
----------  ----------  ----------------  --------------------
Char 0      ASCI0:      RS-232            115200,8,N,1
Char 1      ASCI1:      RS-232            115200,8,N,1
Disk 0      MD0:        RAM Disk          352KB,LBA
Disk 1      MD1:        ROM Disk          384KB,LBA
Disk 2      IDE0:       Hard Disk         122MB,LBA
Disk 3      IDE1:       CompactFlash      244MB,LBA
Disk 4      PPIDE0:     Hard Disk         122104MB,LBA
Disk 5      PPIDE1:     Hard Disk         --
Disk 6      SD0:        SD Card           3781MB,LBA
Disk 7      CHUSB0:     USB Drive         499MB,LBA

Small Computer SC126 [SCZ180_sc126_std] Boot Loader

Boot [H=Help]:
------------------------------------------------------------------------
```

7.2 Laden des RCBD-ROM

RCBD-ROM kann auf zwei Wegen geladen und zur Ausführung gebracht werden.

- Laden über die serielle Konsole unter Kontrolle des *RomWBW*-Monitors. Hierbei liegt Programmcode des *RCBD-ROM* als Binär- oder Intel-HEX-Datei auf dem Hostcomputer, von dem aus *RCBD* über die serielle Konsole bedient wird.
- Laden von einer *UDOS*-Diskette mit dem Bootloader von *RomWBW*. Das Binärfile des *RCBD-ROM* liegt zusammen mit einer Ladeinformation (Boot Info Block) in einem reservierten Bereich einer *UDOS*-Diskette.

7.2.1 Laden des RCBD-ROM über die serielle Konsole

Am Bootprompt von *RomWBW* kann mit dem Kommando `M[onitor]` (Groß- und Kleinschreibung ist nicht relevant) der *RomWBW*-Monitor aufgerufen werden. Er meldet sich mit der aktuellen User-Bank-Nummer als Prompt, hier die Bank '8E'.

```
------------------------------------------------------------------------
Boot [H=Help]: m

Loading Monitor...

Monitor Ready (? for Help)
8E>
------------------------------------------------------------------------
```

Der *RomWBW*-Monitor besitzt das Kommando `L[oad]` zum Laden von Intel-HEX Dateien in den RAM. Die zu ladende Intel-HEX-Datei muss dabei im Terminalprogramm ausgewählt werden. Dazu muss die Konsole über ein Terminalemulationsprogramm benutzt werden, welches eine Funktion zum Senden von Dateien ohne bestimmtes Protokoll besitzt. Bei TeraTerm geschieht das über 'Datei'->'Datei senden'->'Browsen zur Datei'->'Öffnen'->'Ok'. Das *RCBD-ROM*-Image liegt als Intel-HEX-Datei vor.

```
------------------------------------------------------------------------
8E>L

Loaded
8E>
------------------------------------------------------------------------
```

Alternativ kann das Monitorkommando `T[ransfer]` benutzt werden, wenn das zu ladende Programm als Binärdatei vorliegt. Hierbei ist zusätzlich die Angabe der Ladeadresse notwendig. Die zu ladende Binärdatei muss dabei im Terminalprogramm ausgewählt werden. Dazu muss die Konsole über

ein Terminalemulationsprogramm benutzt werden, welches eine Funktion zum Übertragen von Dateien im XMODEM Protokoll besitzt. Bei TeraTerm geschieht das über 'Datei'->'Transfer'->'XMODEM'->'Senden'->'Browsen zur Datei'->'Öffnen'->'Ok'. Das *RCBD-ROM*-Image liegt auch als Binärdatei vor.

```
----------------------------------------------------------------------
8E>T 0

Monitor Ready

Loaded
8E>
----------------------------------------------------------------------
```

Nachdem *RCBD-ROM* fehlerfrei in den RAM des *RomWBW*-Systems geladen wurde, wird es mit `R[un]` an der Adresse 0 gestartet. *RCBD-ROM* führt interne Initialisierungen seiner Datenstrukturen durch (wie jedes *UDOS*-System). Außerdem werden die zur Verfügung stehenden Massenspeicher ermittelt und angezeigt. Falls die Hardware mit einer von *RomWBW* unterstützten Echtzeituhr (RTC) ausgestattet ist, wird das *UDOS*-Systemdatum (DATE) mit dem ausgelesenem Datum voreingestellt. Ist keine RTC vorhanden wird DATE mit dem Build-Datum der *RCBD-ROM*-Datei vorbelegt.

```
----------------------------------------------------------------------
8E>R 0
Device 0 - RAM Disk
Device 2 - 0f Slice(s)*
Device 3 - 1e Slice(s)
Device 4 - ff Slice(s)
Device 6 - ff Slice(s)
Device 7 - 3e Slice(s)
Date set from RTC
UDOS for Z180 RomWBW HBIOS 251125 v2
>
----------------------------------------------------------------------
```

Final meldet sich der *RCBD-ROM*-Monitor mit seinem Prompt `'>'`. Das Größerzeichen `'>'` ist der Prompt des *RCBD-ROM*-Monitors/Debuggers.

7.2.2 Starten des RCBD-ROM mit dem RomWBW Loader

Die elegantere Methode, *RCBD-ROM* zu laden und zu starten besteht darin, das binäre *RCBD-ROM*-Image zusammen mit einer *RomWBW* spezifischen Ladestruktur (*RomWBW*-Loader) als bootfähiges Betriebssystem in einem Slice eines Diskdevices abzulegen (*RCBD-ROM*-Loader). Dann kann am Bootprompt von *RomWBW* das *RCBD-ROM* direkt unter Angabe von Device- und Slicenummer geladen und gestartet werden.

Der *RCBD-ROM*-Loader liegt dazu im Normalfall auf reservierten Sektoren einer *UDOS*-Diskette, bevorzugt auf der Systemdiskette des *RCBD-UDOS*-Systems (*RCBD-ROM*-Loader-on-Disk). Prinzipiell kann aber jede *UDOS*-Diskette mit einem Bootloaderbereich ausgestattet werden. Für das Erzeugen einer *UDOS*-Diskette inklusive *RCBD-ROM*-Loader muss die Diskette beim Formatieren durch das **'format'** Dienstprogramm (11.1) speziell vorbereitet werden.

Alternativ kann der *RCBD-ROM*-Loader auf einem separaten Slice platziert werden, auf dem keine weiteren *UDOS*-spezifischen Datenstrukturen vorhanden sind (*RCBD-ROM*-Loader-only-Slice).

```
------------------------------------------------------------------
Boot [H=Help]: 4.11

Booting Disk Unit 4, Slice 11, Sector 0x0002C800...

Volume "RCBD Z180 251125" [0x0000-0x1FFF, entry @ 0x0000]...
Device 0 - RAM Disk
Device 2 - 0f Slice(s)
Device 3 - 1e Slice(s)
Device 4 - ff Slice(s)*
Device 6 - ff Slice(s)
Device 7 - 3e Slice(s)
Date set from RTC
UDOS for Z180 RomWBW HBIOS 251125 v2
>
------------------------------------------------------------------
```

Hier wurde das *RCBD-ROM* von Device 4 (PPIDE0:) Slice Nummer 11 geladen und automatisch gestartet.

7.3 Initialisierungsmeldungen des RCBD-ROM

RCBD analysiert als erstes die *RomWBW*-Umgebung und gibt Fehlermeldungen über Zustände aus, die den Start von *RCBD* verhindern. Das kann eine unsupportete Hardware-Plattform sein, eine falsche *RomWBW*-Version oder ein nicht geeignetes Bank-Layout.

Bei erfolgreicher Analyse werden nach der Initialisierung und vor der ersten Ausgabe des *RCBD-ROM*-Prompts Meldungen über die ermittelten für *UDOS* geeigneten Massenspeichergeräte angezeigt. Dabei werden die Devicenummern (die Nummern korrepondieren mit der Diskdevice Nummerierung beim Starten von *RomWBW*, siehe Punkt 7.1) und die Eigenschaften des Devices angezeigt.

In den Startmeldungen aus Punkt 7.2 bzw. 7.2.2 finden wir als Device 0 die RAM-Disk (unter RomWBW MD0:) und die Harddisk-Devices 2 (IDE0: = CF-Karte), 3 (IDE1: = CF-Karte),

4 (PPIDE0: = Festplatte), 6 (SD0: = SD-Karte) und 7 (CHUSB0: = USB-Stick). Die Angabe der Sliceanzahl (im *RCBD-ROM*-Monitor sind alle Zahlenangaben hexadezimal) gibt Auskunft, wieviel *UDOS* Floppy-Disk-Images auf dem Datenträger untergebracht sein können (hier also 15, 30, 255 bzw. 62 Slices in dezimaler Interpretation). Dabei handelt es sich um die maximale Anzahl von Slices, die in der *RomWBW* Partition des jeweiligen Datenträgers Platz finden. Eine Nutzung einzelner Slices für andere von *RomWBW* bereitgestellte Betriebssysteme (CP/M, BASIC) wird nicht automatisch erkannt und muss durch den Anwender selbst überwacht werden.

Ein `'*'` hinter einem Harddisk-Device kennzeichnet das Standard- oder Default-Harddisk-Device. Dies wird in Abschnitt 9.1.4 beschrieben.

In Abhängigkeit von der Hardwareausstattung mit einer unterstützten Echtzeituhr (RTC) wird noch eine Meldung ausgegeben ob das *UDOS*-Systemdatum von der RTC gesetzt wurde oder ob keine RTC im System vorgefunden wurde.

In Anlage D sind die Startmeldungen für eine Z80-basierte *RCBD*-Implementierung abgebildet.

7.4 Arbeiten im ROM Monitor

An dieser Stelle kann im *RCBD-ROM*-Monitor/Debugger gearbeitet werden. Die Debuggerkommandos entsprechen denen der *RIO*- bzw *UDOS*-Systeme, wurden aber im Laufe der Zeit 'perfektioniert'. Die Übersicht der implementierten Debuggerkommandos ist in Anlage L zu finden.

```
------------------------------------------------------------------------
>d 0 40
0000 f3 c3 de 06 52 63 42 64 c3 8e 0f 21 04 0d 18 03  >....RcBd...!....<
0010 c3 91 0f 22 bd 0f 18 03 c3 94 0f fb 3e 3e 18 03  >..."........>>..<
0020 c3 97 0f cd 6a 04 18 03 c3 9a 0f 21 9b 00 18 03  >....j......!....<
0030 c3 9d 0f e5 cb af 18 03 c3 84 06 21 76 00 01 0e  >...........!v...<
>r
sz.h.pnc a  b  c  d  e  h  l  i  ixiy pcsp  mem
00000000 00 00 00 00 00 00 00 ff 0000 0000 f3 c3 0b 00
00000000 00 00 00 00 00 00 00 01 0000 fcd0 ff ff ff ff
>
------------------------------------------------------------------------
```

7.5 Start von UDOS

Das eigentliche *UDOS*-Betriebssystem wird mit dem Kommando `os` gestartet. Ein zweistufiger Bootloader sucht die Dateien **'OS'** und **'NDOS'** auf der Diskette im Laufwerk 0, lädt sie an die vorgesehenen Adressen im Hauptspeicher und startet das Betriebssystem an.

Aus einer Datei **'os.init'** liest das **OS** weitere *UDOS*-Dienstprogramme und führt sie bei der Initialisierung aus. Dadurch können zusätzliche Systemeinstellungen getroffen oder einfach nur informative Meldungen ausgegeben werden. Nach erfolgreicher Initialisierung meldet sich *UDOS* mit der Eingabeaufforderung `rcbd:`.

```
------------------------------------------------------------------------
>os
rcbdboot: Ok
osload:   Ok

    ************************
          RCBD OS 2.12
               V2
      UDOS auf RomWBW/HBIOS
    ************************

OS successful patched with HighStack Patch for RCBD
Mittwoch, Februar 26, 2025

Drive 0: 5,25" DD, DS,  80 tracks, 32 sectors per 256 byte (5)
Drive 1: 5,25" DD, DS,  80 tracks, 32 sectors per 256 byte (5)
Drive 2: 5,25" DD, DS,  80 tracks, 32 sectors per 256 byte (5)
Drive 3: 5,25" DD, DS,  80 tracks, 32 sectors per 256 byte (5)

TTY/Screen size 137x34

| Drv0 | Drv1 | Drv2 | Drv3 |
| 2.12 |      |      |      |

Drive 0   RCBD-Sys CF2.12 250225
Floppy Disk - Type 5 - 80 tracks  32 sectors
1921 sectors used
 639 sectors avialable
RIO REL 2.12
rcbd:
------------------------------------------------------------------------
```

8 Laufwerkskonfiguration

Standardmäßig bootet *RCBD* mit einer Konfiguration von vier Diskettenlaufwerken vom Typ 5. Das Typkonzept für Laufwerke wurden vom P8000 übernommen und ermöglichte dort die Anpassung an verschiedene, damals auf dem Mark befindliche Diskettenlaufwerke (40/80-spurig, ein- oder doppelseitig), ohne den `$FLOPPY`-Treiber oder geometrieabhängige Dienstprogramme wie **'format'**, **'status'** oder **'copy.disk'** für jeden P8000 individuell generieren zu müssen.

Im *RCBD* wird derzeit nur der Laufwerkstyp 5 (80 Spuren, doppelseitig, 32 Sektoren pro Spur) unterstützt. Für das RAM-Disk-Laufwerk wurden im *RCBD* die Typen 9 und A (hexadezimale Ziffer A) hinzugefügt. Alle übrigen vom P8000 bekannten Laufwerkstypen sind prinzipiell einstellbar, werden aber vom `$FLOPPY`-Treiber nicht behandelt. Ein Zugriff auf Laufwerke, die nicht als Typ 5, Typ 9 oder Typ A eingestellt sind, wird mit einem 'Drive not ready' Fehler (Error C2) beantwortet.

8.1 Dienstprogramm 'setfd'

Mit **'setfd'** lässt sich die Laufwerkskonfiguration anzeigen oder ändern.

```
------------------------------------------------------------------------
rcbd:setfd
Drive 0: 5,25" DD, DS,  80 tracks, 32 sectors per 256 byte (5)
Drive 1: 5,25" DD, DS,  80 tracks, 32 sectors per 256 byte (5)
Drive 2: 5,25" DD, DS,  80 tracks, 32 sectors per 256 byte (5)
Drive 3: 5,25" DD, DS,  80 tracks, 32 sectors per 256 byte (5)
------------------------------------------------------------------------
```

Grundzustand mit vier Diskettenlaufwerken Typ 5.

```
------------------------------------------------------------------------
rcbd:setfd 5955
Drive 0: 5,25" DD, DS,  80 tracks, 32 sectors per 256 byte (5)
Drive 1: RAM Drive 256k 32 tracks, 32 sectors per 256 byte (9)
Drive 2: 5,25" DD, DS,  80 tracks, 32 sectors per 256 byte (5)
Drive 3: 5,25" DD, DS,  80 tracks, 32 sectors per 256 byte (5)
------------------------------------------------------------------------
```

Laufwerk 1 wurde als RAM-Disk-Laufwerk (Typ 9) konfiguriert. Die Laufwerke 0, 2 und 3 sind weiterhin Diskettenlaufwerke vom Typ 5.

Es kann jedes der vier Laufwerke als RAM-Disk-Laufwerk konfiguriert werden, und auch mehrere gleichzeitig. Da *RomWBW* aber nur ein RAM-Disk-Device bereit stellt, macht die gleichzeitige Konfiguration von mehr als einem RAM-Disk-Laufwerk unter *UDOS* keinen Sinn.

Eine Liste der mit **'setfd'** einstellbaren Laufwerkstypen ist in Anlage I zu finden. Im *RCBD* kann die Liste der definierten Laufwerkstypen mit `setfd l[ist]` angezeigt werden.

Wird als Laufwerkstyp `0` angegeben, kennzeichnet das ein nicht vorhandenes Laufwerk. Ein Zugriff darauf wird mit einem 'Drive not ready' Fehler (Error C2) beantwortet.

```
------------------------------------------------------------------------
rcbd:setfd 5900
Drive 0: 5,25" DD, DS,  80 tracks, 32 sectors per 256 byte (5)
Drive 1: RAM Drive 256k 32 tracks, 32 sectors per 256 byte (9)
Drive 2: No drive
Drive 3: No drive
------------------------------------------------------------------------
```

9 Diskmapping

Das Diskmapping ist ein zentraler Bestandteil des *RCBD*. Ein *UDOS*-Rechner besitzt zwischen ein und vier Diskettenlaufwerke. Der Anwender kann in diese Laufwerke Disketten einlegen und wieder entnehmen. Außerdem können die Disketten beim Einlegen einen Schreibschutz besitzen oder beschreibbar sein.

Im *RCBD* sind die Disketten Images, die in Slices eines oder mehrerer von *RomWBW* unterstützten Speichergeräten (Devices) abgelegt sind. Das Pedant zum Einlegen einer Diskette in ein *UDOS*-Laufwerk ist im *RCBD* das Zuordnen einer Device:Slice Kombination, die ein gültiges *UDOS*-Diskettenimage enthält, zu einer *UDOS*-Laufwerksnummer. Dieser Vorgang wird als 'Diskmapping' bezeichnet.

Natürlich kann auch ein unbenutzter Slice einem *UDOS*-Laufwerk zugeordnet und dann unter *UDOS* als neue Diskette formatiert werden.

Der entgegengesetzte Vorgang, das Aufheben einer Zuordnung zwischen Device:Slice und *UDOS*-Laufwerk entspricht dann dem Entnehmen einer Diskette aus einem physischen Laufwerk.

9.1 Dienstprogramm 'md'

Für diesen Vorgang wurde das Dienstprogramm **'md'**, welches auf der *RCBD*-Systemdiskette zu finden ist, geschrieben. **'md'** ist für das Einlegen, Entnehmen und Steuern des Schreibschutzes von Disketten zuständig. Auch das Einbinden bzw. Abhängen der RAM-Disk wird von **'md'** übernommen.

9.1.1 Diskmapping anzeigen

'md' ohne Argumente zeigt das aktuelle Diskmapping an. Ein einzelnes Gleichheitszeichen `'='` als Argument hat die gleiche Wirkung.

```
-------------------------------------------------------------------
rcbd:md
| Drv0 | Ram1 | Drv2 | Drv3 |
| 2.12 | 0:00 |      | 4:14 |
-------------------------------------------------------------------
```

Die Kopfzeile ist selbsterklärend, vier Diskettenlaufwerke, davon ist Laufwerk 1 ein RAM-Disk-Laufwerk. Diese Konstellation wurde, wie im Kapitel 8 beschrieben, vorher mit dem Dienstprogramm **'setfd'** eingestellt. Ein leeres Feld unter dem Laufwerksbezeichner kennzeichnet ein leeres, ungemapptes Laufwerk. Einen gezielten Zugriff darauf würde *UDOS* mit einem 'Drive not ready' Fehler (Error C2) beantworten. Gemappte Laufwerke werden durch `device.slice` oder `device:slice` angezeigt.

Das Trennzeichen zwischen `device` und `slice`, Punkt oder Doppelpunkt, zeigt einen vorhandenen Schreibschutz an. Als Eselsbrücke kann man sich merken: 'Ein Punkt: Lesen, Zwei Punkte: Lesen und Schreiben'. In Laufwerk 0 steckt also die schreibgeschützte Diskette aus Slice 12 von Device 2. In Laufwerk 3 liegt eine beschreibbare Diskette, deren Image in Slice 14 auf Device 4 gespeichert ist. Das RAM-Disk-Device von *RomWBW* (Device 0) ist beschreibbar im Laufwerk 1 gemappt. Bei der RAM-Disk wird immer Slice 0 angezeigt, die von *RomWBW* bereitgestellte RAM-Disk ist monolithisch und nicht weiter unterteilt.

Den Versuch, auf eine schreibgeschützt gemappte Diskette zu schreiben, quittiert *UDOS* mit einem 'Write protect' Fehler (Error C3)

9.1.2 Diskmapping aufheben

Das Aufheben des Diskmappings, also das Entnehmen einer Diskette, führt **'md'** durch, wenn als Argument die *UDOS*-Laufwerksnummer, gefolgt von einem Gleichheitszeichen `'='`, angegeben wird. Das Resultat wird anschließend angezeigt. Ausgehend vom im Punkt 9.1.1 abgebildeten Status:

```
-----------------------------------------------------------------
rcbd:md 3=
| Drv0 | Ram1 | Drv2 | Drv3 |
| 2.12 | 0:00 |      |      |
-----------------------------------------------------------------
```

Die Diskette aus Laufwerk 3 wurde entnommen.

9.1.3 Diskmapping erstellen

Das Herstellen eines Diskmappings, also das Einlegen einer Diskette, wird durch ein dem Gleichheitszeichen folgendes `device.slice` oder `device:slice` Paar initiiert. Dabei würde ein Punkt `'.'` als Trennzeichen die Diskette schreibgeschützt einlegen, ein Doppelpunkt `':'` würde eine beschreibbare Diskette dem Laufwerk zuordnen. Das Resultat wird anschließend angezeigt. Ausgehend vom im Punkt 9.1.2 abgebildeten Status:

```
-----------------------------------------------------------------
rcbd:md 3=2.4
| Drv0 | Ram1 | Drv2 | Drv3 |
| 2.12 | 0:00 |      | 2.04 |
rcbd:md 2=4:55
| Drv0 | Ram1 | Drv2 | Drv3 |
| 2.12 | 0:00 | 4:55 | 2.04 |
-----------------------------------------------------------------
```

Es wurden zwei weitere Disketten eingelegt, davon eine schreibgeschützt.

RCBD achtet darauf, dass die gleiche Diskette nicht gleichzeitig in mehrere *UDOS*-Laufwerke eingelegt wird, und dass eine Diskette nur in ein leeres

Laufwerk eingelegt werden kann. Das entspricht der natürlichen Arbeitsweise am physischen Rechner - bevor man eine Diskette in ein bestimmtes Laufwerk einlegt, muss man eine andere darin befindliche Diskette entnehmen. Außerdem werden Laufwerks-, Device- und Slicenummern auf Gültigkeit und Existenz überprüft.

9.1.4 Default-Harddisk-Device

Das Default-Harddisk-Device (kurz: DefHDD) stellt eine Vereinfachung beim Diskmapping dar, indem beim Mapping die Devicenummer weggelassen werden kann, wenn sich das Slice mit der *UDOS*-Diskette auf dem Harddisk-Device befindet, dessen Nummer als DefHDD eingestellt ist.

Ist z. B. das DefHDD auf Device 3 eingestellt, sind folgende Kommandos gleichwertig:

```
-----------------------------------------------------------------------
rcbd:md *                                ; Aktuelles DefHHD anzeigen
Default HD device: 03
rcbd:md 3=3.4                            ; Mapping Slice 4 mit Angabe
| Drv0 | Ram1 | Drv2 | Drv3 |           ;            der Devicenummer
| 2.12 | 0:00 |      | 3.04 |
                                         ; ist identisch zu

rcbd:md 3=.4                             ; Mapping Slice 4 des DefHDD
| Drv0 | Ram1 | Drv2 | Drv3 |
| 2.12 | 0:00 |      | 3.04 |
-----------------------------------------------------------------------
```

Die Nutzung des DefHDD rentiert sich besonders, wenn während der Arbeit mit *UDOS* häufig Disketten gewechselt werden müssen und alle (oder die meisten davon) auf dem gleichen *RomWBW*-Datenträger liegen.

Die initiale Einstellung des DefHDD erfolgt während der Initialisierung des *RCBD-ROM*. Das DefHDD wird auf das Device eingestellt, von dem das *RCBD-ROM* geladen wurde. Kann kein Ladedevice ermittelt werden (weil z. B. das *RCBD-ROM*-Image seriell über den *RomWBW*-Monitor in den RAM des Systems geladen und dort gestartet wurde (Punkt 7.2)), wird als Fallback das während der *RCBD-ROM*-Initialisierung gefundene Harddisk-Device mit der niedrigsten Devicenummer als DefHDD eingestellt.

Das DefHDD kann zur Laufzeit des *RCBD-UDOS* jederzeit mit dem Dienstprogramm **'md'** auf ein anderes Harddisk-Device umgestellt werden, welches bei der *RCBD-ROM*-Initialisierung gefunden wurde. Die Angabe einer ungültigen Devicenummer wird mit einer Fehlermeldung quittiert und das bisherige DefHDD wird unverändert beibehalten.

```
-----------------------------------------------------------------
rcbd:md *                       ; Aktuelles DefHHD anzeigen
Default HD device: 03
rcbd:md 2*                      ; Anderes Device als DefHDD einstellen
Default HD device: 02
rcbd:md 0*                      ; Ungültiges Device (Keine HD)
Default HD device not changed
-----------------------------------------------------------------
```

Die in der aktuellen Version von **'md'** enthaltene Option zur Änderung des DefHDD erfordert das *RCBD-ROM* 250816 V2 oder neuer. Bei *RCBD-ROM* 250528 V2 kann das DefHDD angezeigt, aber nicht geändert werden.

9.1.5 Fehlermeldungen von 'md'

'md' kann folgende Fehlermeldungenen ausgeben:

```
Invalid UDOS drive number
Unsupported HBIOS device number
Unusable HBIOS device (no HD in hd1k format or no RAM disk)
Invalid slice number for device
Drive not empty (a different slice is already mapped)
Slice already mapped to another drive
Drive and device incompatible (FD in RAM drive or so)
Default HD device not changed
```

Eine detailierte Beschreibung der Fehlermeldung finden Sie in Anlage G.4.

9.1.6 Kombinieren mehrer Mappings

Beim Aufruf von **'md'** können mehrere Map- und Unmap-Anweisungen in einer Kommandozeile angegeben werden. Das vereinfacht den Diskettenwechsel, da dann das Entnehmen der bisherigen Diskette und das Einlegen einer neuen in das gleiche Laufwerk mit einem einzigen Aufruf von **'md'** erledigt werden kann. Es können alle syntaktisch richtigen Argumente in einer Kommandozeile kombiniert werden. Das gilt auch für das Ändern des defHDD sowie die reinen Anzeigeoptionen von **'md'** für das aktuelle Mapping `'='` und das aktuelle DefHDD `'*'`.

```
-----------------------------------------------------------------
rcbd:md
| Drv0 | Ram1 | Drv2 | Drv3 |
| 2.12 | 0:00 |      | 4:14 |
rcbd:md 3= 2* 3=.4 2=4:55
| Drv0 | Ram1 | Drv2 | Drv3 |
| 2.12 | 0:00 |      |      |
Default HD device: 02
| Drv0 | Ram1 | Drv2 | Drv3 |
| 2.12 | 0:00 |      | 2.04 |
| Drv0 | Ram1 | Drv2 | Drv3 |
```

```
| 2.12 | 0:00 | 4:55 | 2.04 |
-----------------------------------------------------------------
```

Die Beispiele aus den Punkten 9.1.2 und 9.1.3 hier in einer einzigen Kommandozeile.

Sollte es bei einem der aneinander gereihten Mappings zu einem Fehler kommen, wird der Fehler entsprechend Punkt 9.1.5 bzw. Anlage G.4 angezeigt und die Abarbeitung der Kommandozeile abgebrochen. Die restlichen Mappings in der Kommandozeile werden dann nicht mehr ausgeführt.

9.1.7 Nutzung der RAM-Disk

Startet *RCBD* nach einem Power On, hat der RAM-Chip, der die RAM-Disk bereitstellt, undefinierte Daten. Nach dem Mappen des RAM-Disk-Devices in das als RAM-Drive konfigurierte Laufwerk (hier Laufwerk 1) muss die RAM-Disk formatiert werden, bevor sie als *UDOS*-Datenträger nutzbar ist. Es stehen derzeit zwei RAM-Disk-Größen zur Auswahl, 256 KiB (Laufwerkstyp 9) und 352 KiB (Laufwerkstyp A), siehe Anlage I. Die hardwareseitigen Voraussetzungen für die beiden Format sind in Anlage J beschrieben.

```
-----------------------------------------------------------------
rcbd:md
| Drv0 | Ram1 | Drv2 | Drv3 |
| 2.12 |      |      |      |
rcbd:md 1=0:0
| Drv0 | Ram1 | Drv2 | Drv3 |
| 2.12 | 0:00 |      |      |
rcbd:format d=1 id='Temporary RAM-Disk 256k' q=n
RCBD Disk Formatter 250528
rcbd:status

Drive 0   RCBD-Sys CF2.12 250225
Floppy Disk - Type 5 - 80 tracks  32 sectors
1858 sectors used
 702 sectors avialable

Drive 1   Temporary RAM-Disk 256k
   RAM Disk - Type 9 - 32 tracks  32 sectors
   9 sectors used
1015 sectors avialable
-----------------------------------------------------------------
```

Die Formatierung der RAM-Disk und deren Inhalt bleiben erhalten, solange die *RCBD*-Hardware nicht ausgeschaltet wird oder im Falle eines Resets (Taste) vor dem nächsten Start des *RCBD* kein anderes, von *RomWBW* bereitgestelltes Betriebssystem (z. B. CP/M) gebootet wird, welches die *UDOS*-Formatierung der RAM-Disk zerstört.

Folgende Aktivitäten verändern nicht den Inhalt der RAM-Disk:

- Unmappen und späteres neu Mappen der RAM-Disk
- Aufruf des *RCBD-ROM*-Monitors/Debuggers und Initiieren eines neuen *UDOS*-Bootvorgangs (`os` Kommando im Debugger)
- Aufruf des *RCBD-ROM*-Monitors/Debuggers und Rückkehr zum *HBIOS*-Bootprompt (`qx` Kommando im Debugger), anschließendes Laden des *RCBD-ROM* und Booten von *UDOS*
- Hardware Reset (Taste) ohne zwischenzeitliches Power Off, Laden des *RCBD-ROM* und Booten von *UDOS*

9.1.8 Initiales Mapping

Um das *UDOS* starten zu können, benötigt man ein initiales Diskmapping für das Laufwerk 0 auf ein Slice mit dem Image einer *RCBD*-Systemdiskette. Beim Start des *RCBD-ROM* wird daher dem Laufwerk 0 ein Standardslice als schreibgeschützte Diskette zugeordnet. Die Zuordnung basiert auf einer Designentscheidung, die ich getroffen habe und deren Umsetzung im *RCBD-ROM* kodiert ist.

- In der Initialisierungsphase des *RCBD-ROM* werden alle für *RCBD* nutzbaren *RomWBW*-Devices ermittelt. Das Ergebnis sieht dann z. B. so aus, wie in Punkt 7.2.2 gezeigt und in Punkt 7.3 beschrieben.
- Die Systemdiskette wird in dem Harddisk-Device erwartet, welches während der *RCBD-ROM*-Initialisierung als Default-Harddisk-Device eingestellt wurde, im abgebildeten Beispiel also im Device 2.
- Bei Datenträgern mit weniger als 15 Slices wird die Systemdiskette in Slice 0 des Devices erwartet.
- Bei Datenträgern mit 15 oder mehr Slices muss das Image der Systemdiskette in Slice 12 abgelegt werden.

Hintergrund dieser Festlegung ist, dass bei Medien (CF-/SD-Karten etc.) höherer Kapazität die unteren Slices frei bleiben für die von *RomWBW* serienmäßig enthaltenen Betriebssystemimages von CP/M, BASIC, Forth etc.. Nutzt man diese Betriebssysteme nicht, können natürlich die Slices unterhalb von 12 für *UDOS*-Disketten genutzt werden. Nur die Position der Systemdiskette in Slice 12 ist hart kodiert. Siehe Anlage C für typische Datenträgergrößen.

9.1.9 Entnahme der Systemdiskette

Was tun, wenn man die Systemdiskette entnommen hat und dadurch des **'md'** Dienstprogramm nicht mehr zur Verfügung steht?

```
------------------------------------------------------------------------
rcbd:md 0=
| Drv0 | Ram1 | Drv2 | Drv3 |
|      | 0:00 |      |      |
------------------------------------------------------------------------
```

Dazu gibt es mehrere Lösungsansätze. Den *RCBD* neu starten ist davon der schlechteste.

Variante 1 ist, das **'md'** Dienstprogramm auf alle Disketten, mit denen man arbeitet, zu kopieren. *UDOS* durchsucht beim Aufruf eines externen Programms alle bereiten Laufwerke in aufsteigender Reihenfolge, bis es das gesuchte Programm gefunden hat. Solange also irgendeine entsprechend vorbereitete Diskette in einem Laufwerk liegt, steht **'md'** zum erneuten Mappen der Systemdiskette ins Laufwerk 0 zur Verfügung.

Man könnte also bespielsweise **'md'** zu Beginn der Arbeit gleich auf die frisch formatierte RAM-Disk kopieren, die man während des gesamten Arbeitszyklus im Allgemeinen nicht entnimmt.

```
------------------------------------------------------------------------
rcbd:md
| Drv0 | Ram1 | Drv2 | Drv3 |
| 2.12 |      |      |      |
rcbd:md 1=0:0
| Drv0 | Ram1 | Drv2 | Drv3 |
| 2.12 | 0:00 |      |      |
rcbd:format d=1 id='Temporary RAM-Disk 256k' q=n
RCBD Disk Formatter 250224
rcbd:copy 0/md 1/md o
------------------------------------------------------------------------
```

Variante 2 nutzt die Eigenschaft des **OS** aus, dass ein ausgeführtes Programm nach Beendigung im Speicher verbleibt und neu angestartet werden kann, solange der Speicherbereich nicht durch ein anderes Programm überschrieben wurde. Dazu wird das interne **OS** Kommando `x[ecute]` benutzt.

```
------------------------------------------------------------------------
rcbd:md 0=
| Drv0 | Ram1 | Drv2 | Drv3 |
|      |      |      |      |
rcbd:md 0=2.12
Nonexistent command
rcbd:x * 0=2.12
| Drv0 | Ram1 | Drv2 | Drv3 |
| 2.12 |      |      |      |
------------------------------------------------------------------------
```

Variante 3: Im *RCBD-ROM*-Monitor/Debugger wurde ebenfalls ein Kommando `md` implementiert, auf das zurückgegriffen werden kann. Dies ist funktionell identisch zum externen **'md'** Dienstprogramm, benutzt aber eine ver-

einfachte Syntax (bei nur 3 KiB Platz im residenten Teil muss man Abstriche an die Schönheit machen).

- Alle Zahlenangaben (*UDOS*-Laufwerksnummer, Device- und Slicenummern) sind hexadezimal einzugeben bzw. werden in Hex angezeigt.
- Der Schreibschutz einer Diskette wird aufgehoben, indem beim Mappen die Laufwerksnummer mit gesetztem Bit 7 notiert wird.
- Device- und Slicenummer sind als 16-Bit-Zahl zusammengefasst, wobei das MSB die Devicenummer enthält und das LSB die Slicenummer.

In der Kommandoreferenz des *RCBD-ROM*-Monitors (Anlage L.2) steht die ausführliche Syntaxbeschreibung. Die Beispiele aus den Punkten 9.1.1 bis 9.1.3 hier mit Kurzkommentaren in der `md`-Variante aus dem *RCBD-ROM*-Monitor.

```
---------------------------------------------------------------------
rcbd:md                                 ; Ausgangssituation
| Drv0 | Ram1 | Drv2 | Drv3 |
| 2.12 | 0:00 |      | 4:14 |
rcbd:d                                  ; Aufruf ROM-Monitor/Debugger
>md                                     ; Mapping anzeigen
2.0c 0:00 *.ff 4:0e
>md 3                                   ; Mapping LW 3 löschen
2.0c 0:00 *.ff *.ff
>md 3 204                               ; Mapping LW 3 Read only
RC = 00
2.0c 0:00 *.ff 2.04
>md 82 437                              ; Mapping LW 2 Read/Write
RC = 00
2.0c 0:00 4:37 2.04
>q                                      ; Rückkehr ins OS
rcbd:md                                 ; Aktueller Zustand
| Drv0 | Ram1 | Drv2 | Drv3 |
| 2.12 | 0:00 | 4:55 | 2.04 |
---------------------------------------------------------------------
```

10 RCBD spezifische UDOS Kernkomponenten

10.1 RCBD-ROM

RCBD-ROM ist die residente Komponente des *RCBD*. Sie entspricht funktionell dem 3 KiB ROM der Zilog Systeme bzw. den 3 KiB residenter Software, die bei den DDR *UDOS*-Implementierungen von speziell reservierten Sektoren der Systemdiskette oder einem versteckten ROM in den unteren Teil des Hauptspeichers geladen wurde. Der Code von *RCBD-ROM* belegt den Speicherbereich `0x0000..0x0BFF`, zusätzlich wird 1 KiB operativer Speicher im Bereich `0x0C00..0x0FFF` benutzt. Dieser operative Speicher (System-RAM) wird nach dem Booten von *UDOS* auch von den Komponenten **OS** und **NDOS** benutzt.

Die *RCBD-ROM*-Imagedatei enthält weiteren Code, der zur Initialisierung dient und nur beim ersten Start des *RCBD-ROM* nach dem Laden einmalig ausgeführt wird. Dieser liegt ab Adresse `0x1000` im Hauptspeicher und wird anschließend durch das Booten von *UDOS* überschrieben.

Ein dritter Codeabschnitt in der *RCBD-ROM*-Imagedatei enthält Code, der während der *RCBD-ROM*-Initialisierung in eine andere RAM-Bank (bei mir Shadow-Bank genannt) kopiert wird und für *UDOS* und seine Dienst- und Anwendungsprogramme unsichtbar ist. Diese Programmteile enthalten den Code, der nicht *UDOS*-typisch ist, aber für die Interaktion zwischen *RCBD-ROM* und der *HBIOS*-Schnittstelle gebraucht wird. Diese Routinen werden aus dem *RCBD-ROM* über *HBIOS*-Aufrufe angesprochen, z. B. innerhalb des `$FLOPPY`-Treibers.

Nach Initialisierung des *RCBD-ROM* verbleibt lediglich der ROM-Bereich `0x0000..0x0BFF` sowie der System-RAM `0x0C00..0x0FFF` im 64 KiB-Adressraum des *RCBD*-Systems.

10.2 OS

Die Datei **'OS'** wurde auf Byteebene minimal gepatcht. Zum einen benutzt das **OS** als Kernkomponente von *UDOS* einen statischen Stack von 128 B, der innerhalb seiner Programmstruktur `0x1000..0x20FF` und somit in der User-Bank unterhalb von `0x8000` liegt. Dieser Stack wurde in die Common-Bank verlegt.

UDOS Dienst- und Anwendungsprogramme fordern beim Laden dynamisch einen Stack an, dessen Größe in der Dateieigenschaft `stack_size` festgelegt ist und die ein ganzzahliges Vielfaches von 128 B (`0x80`) beträgt. Die originalen Routinen im **OS** weisen als Stack einen Block der angeforderten Größe am unteren Ende des größten zusammenhängenden freien Hauptspeicherblocks zu. Dieser Bereich liegt gewöhnlich unterhalb der Adresse `0x8000`

und somit in der User-Bank. Die Änderung im **OS** sorgt dafür, dass dynamisch zugewiesener Stack garantiert in der Common-Bank, also oberhalb der Adresse `0x8000` liegt.

10.3 NDOS

Der Treiber `$NDOS` benötigt auf Grund der komplexeren Dateiverwaltung gegenüber dem `$ZDOS` von *RIO* und früheren *UDOS*-Versionen mehr Speicher und zusätzliche Puffer. Diese liegen ursprünglich (beim P8000) am oberen Ende des Hauptspeichers und belegen 1280 B (`0x500`). Da im *RCBD* am oberen Ende das Hauptspeichers der *HBIOS*-Proxy und die umgelegten statischen Stacks liegen, wurde dieser Bereich um 1280 B nach unten verlagert und belegt jetzt den Adressraum `0xF600..0xFAFF`.

10.4 PATCHOS

```
Usage: PATCHOS [-R]
```

'PATCHOS' ist ein residentes Programm, welches den Algorithmus im **OS** zur Vergabe dynamischen Stacks an aufgerufene externe Programme im oberen Speicherbereich weiter optimiert. **'PATCHOS'** kann zur Laufzeit von *UDOS* geladen oder entladen werden. **'PATCHOS'** lässt sich gut in der Datei **'os.init'** unterbringen, so dass es beim Booten von *UDOS* automatisch geladen wird.

```
-----------------------------------------------------------------------
rcbd:PATCHOS                                                  ; Laden
OS successful patched with HighStack Patch for RCBD
rcbd:PATCHOS -R                                               ; Entladen
HighStack Patch for RCBD removed from OS
-----------------------------------------------------------------------
```

Anlage G.3 enthält die vollständige Liste aller Meldungen, die **'PATCHOS'** erzeugen kann.

10.5 $PRTC

`$PRTC` ist ein zusätzlicher Treiber ohne Vorbild im P8000 *UDOS*, der den *UDOS*-konformen Zugriff von Anwendungsprogrammen auf die Hardware einer von *RomWBW* unterstützten Echtzeituhr (RTC) ermöglicht. Der Treiber ist ROM-resident im *RCBD-ROM* untergebracht.

In Anlage M sind Aufruf- und Rückgabeparameter des Treibers beschrieben.

11 Neue und modifizierte Dienstprogramme

UDOS Dienst- und Anwendungsprogramme werden, soweit sie nicht in diesem Kapitel explizit beschrieben werden, wie in der originalen *RIO*- bzw. *UDOS*-Dokumentation benutzt.

Im *UDOS*-Dateisystem sind Dateinamen case sensitiv. D. h. dass externe Dienst- und Anwendungsprogramme, die als Datei von der Diskette geladen werden, in der Schreibweise aufgerufen werden müssen, in der sie auf der Diskette gespeichert sind. Ich habe (bereits in den 80er Jahren) auf meinen *UDOS*-Rechnern alle interaktiv aufgerufenen Dateien auf den Systemdisketten in Kleinschreibung umbenannt. Das gefiel mir besser und war vom Tastaturhandling 'natürlicher'. Bestimmte Dateien, die nicht interaktiv, sondern als Overlay von anderen Dateien aufgerufen werden, habe ich in Großschreibung belassen. So ruft beispielsweise der Z80 Assembler **'ASM'** (auf meinen Systemdisketten **'asm'**) die Overlaydateien **'ASM2'** und **'ASM3'** auf, die weiterhin in Großschreibung auf den Disketten gespeichert sind.

11.1 format

```
Usage: format [d=<drive>] [id='<max_24_char'] [q=n] [s] [ll=y] [rl=y]
```

'format' ist das Formatierungsprogramm für Disketten in *RIO* und *UDOS*. Für den *RCBD* wurde dieses Programm dahingehend modifiziert, dass beim Formatieren einer Diskette keine Lowlevel-Formatierung erfolgt. Das ist einerseits technisch nicht notwendig, da es sich im *RCBD* nicht um magnetische Datenträger sondern um Speicherabbilder (Images) handelt. Zum anderen ist es natürlich sinnvoll, die flashbasierten Datenträger, die die Images der Disketten enthalten (CF-/SD-Karten etc.), nicht unnötig mit Schreiboperationen zu belasten, da dies ihre Lebensdauer reduziert.

'format' auf dem *RCBD* initialisiert daher lediglich die Disk-Allocation-Map, legt ein leeres Inhaltsverzeichnis (**'DIRECTORY'**) an und kopiert optional die Systemsektoren (*UDOS*-Bootloader), wenn die Option zum Formatieren einer Systemdiskette angegeben wurde.

Eine Pseudo-Lowlevel-Formatierung (Beschreiben alle Sektoren der Diskette mit dem Füllbyte `0xE5`) kann durch eine neue Kommandozeilenoption `ll=y (LowLevel=Yes)` erzwungen werden. Die Option wird ignoriert, wenn es sich bei der zu formatierenden Diskette um eine RAM-Disk handelt.

Mit der neuen Option `rl=y (ROM_Loader=Yes)` kann während des Formatierens der Sektorbereich für den *RCBD-ROM*-Loader (Boot-Info-Block plus *RCBD-ROM*-Datei) in der Disk-Allocation-Map reserviert werden. Der *RCBD-ROM*-Loader-Code wird dabei nicht durch das **'format'** Programm auf die Diskette kopiert, es wird lediglich der notwendige Sektorbereich als belegt gekennzeichnet, damit die regulären Dateioperationen von *UDOS* dort

keine Dateien ablegen können. Der *RCBD-ROM*-Loader-Bereich verringert die nutzbare Diskettenkapazität um 34 Sektoren.

Die Option `rl=m (ROM_Loader=Mini)` reserviert ähnlich der `rl=y` Option Sektoren für einen *RomWBW*-Loader. Die Größe des reservierten Bereiches beträgt allerdings nur 3 Sektoren (Boot-Info-Block plus 256 Byte Imagedatei). Diese Option ist für die Nutzung einer zukünftigen Funktion in *RomWBW* Version 3.6 vorgesehen.

Verschiedene Verfahren, das *RCBD-ROM*-Loader-Image in die reservierten Sektoren zu schreiben, sind beim Dienstprogramm **'copy.disk'** (11.5), **'romwrite'** (11.6) bzw. in den Anlagen O.4 und O.5 beschrieben.

Alle anderen Optionen von **'format'** arbeiten wie in der originalen Programmversion. Die von **'format'** initial reservierten Sektoren für die verschiedenen Formatierungsoptionen zeigen die Tabellen in Anlage K.

11.2 status

```
Usage: status [<drive>]
```

'status' zeigt den Status eines bestimmten oder aller bereiten Diskettenlaufwerke und die Belegungsstatistik der eingelegten Diskette(n) an. Für den *RCBD* wurde **'status'** um die Ausgabe des Status der RAM-Disk erweitert.

11.3 md

```
Usage: md [-v]
       md [<device>]*
       md [<drv>]=
       md <drv>=[<device>]{.|:}<slice>
```

Dieses neue Dienstprogramm wurde bereits in Kapitel 9 beschrieben.

11.4 setfd

```
Usage: setfd [<typ0><typ1><typ2><typ3> [s<drv>]]
       setfd l[ist]
```

Dieses Dienstprogramm wurde bereits in Kapitel 8 beschrieben.

11.5 copy.disk

```
Usage: copy.disk s=<src_drv> d=<dst_drv> [b|r [v]] [q=n]
```

'copy.disk' kopiert in seiner Grundfunktion ohne die Optionen `b` oder `r` komplette *UDOS*-Disketten (natürlich deren Images) sektorweise von einem

Laufwerk auf ein anderes. Es kann nur zwischen Laufenwerken gleichen Formats (gleichen Laufwerktyps) kopiert werden. Ein Kopieren zwischen Diskettenlaufwerk und RAM-Disk-Laufwerk ist auf Grund der unterschiedlichen Geometrien nicht möglich.

Quelllaufwerk `src_drv` kann jedes der vier Diskettenlaufwerke sein, als Ziellaufwerk `dst_drv` können nur die Laufwerke 1 bis 3 dienen. Das schützt die Systemdiskette in Laufwerk 0 vor Überschreiben, falls diese nicht schreibgeschützt gemappt ist.

Die Diskette im Quelllaufwerk kann schreibgeschützt gemappt sein. Im Ziellaufwerk muss die Diskette beschreibbar gemappt sein. Im Ziellaufwerk muss nicht unbedingt eine formatierte Diskette gemappt sein, auch ein leerer oder von Fremdsystemen befüllter Slice würde mit der Kopie der Diskette aus dem Quelllaufwerk überschrieben werden.

Sofern nicht die Option `q=n` angegeben wurde, muss jede Kopieroperation mit `y[es]` bestätigt werden, alle anderen Tasten brechen das Programm ab.

```
----------------------------------------------------------------------
rcbd:copy.disk s=1 d=3
copy.disk 251204
Copy the whole disk in drive 1 to drive 3 [Ny]?
----------------------------------------------------------------------
```

Mit **'copy.disk'** können Sicherheitskopien ganzer Disketten gemacht werden, idealerweise auf ein Slice in einem anderen Device.

Mit der `b` Option wird nicht die gesamte Diskette kopiert, sondern nur der primäre und der sekundäre Bootloader einer Systemdiskette. Es wird überprüft, ob im Quelllaufwerk eine Systemdiskette mit einem *RCBD*-Bootloader liegt. Es wird ebenfalls überprüft, ob im Ziellaufwerk eine formatierte Systemdiskette liegt, dessen Bootloader aktualisiert werden soll. Ist im Ziellaufwerk eine Nicht-Systemdiskette oder ein unformatiertes Slice eingelegt, bricht das Programm den Vorgang ab.

```
----------------------------------------------------------------------
rcbd:copy.disk s=0 d=2 b
copy.disk 251204
Copy UDOS boot loader from drive 0 to drive 2 [Ny]?
----------------------------------------------------------------------
```

Mit der `r` Option kann man *RCBD-ROM*-Loader-Images von einem Datenträger auf einen anderen kopieren. Es wird überprüft, ob im Quelllaufwerk ein Slice mit einem gültigen *RCBD-ROM*-Loader-Image gemappt ist (*RCBD-ROM*-Loader-only-Slice oder *RCBD-ROM*-Loader-on-Disk). Es wird nicht überprüft, ob im Ziellaufwerk eine *RCBD-UDOS*-Diskette mit reservierten *RCBD-ROM*-Loader Bereich gemappt ist. Es kommt zu Datenverlust wenn im Ziellaufwerk eine reguläre *UDOS*-Diskette ohne reservierten *RCBD-ROM*-Loader Bereich liegt oder wenn das Slice eines Nicht-*UDOS*-Systems im Ziellaufwerk gemappt ist.

```
----------------------------------------------------------------------
rcbd:copy.disk s=3 d=2 r
copy.disk 251204
Be sure the destination disk has reserved sectors for RCBD ROM loade
Copy RCBD ROM loader from drive 3 to drive 2 [Ny]?
----------------------------------------------------------------------
```

Die `b` und `r` Optionen schließen sich gegenseitig aus. Werden in einer Kommandozeilen beide Optionen gleichzeitig eingegeben, hat die am weitesten rechts notierte den Vorrang. Im Zusammenhang mit den `b` und `r` Optionen gibt die `v` Option zusätzliche Informationen während des Kopiervorgangs aus.

11.6 romwrite

```
Usage: romwrite [<file>] d=<drive> [l=<string>] [v [v]]
       romwrite [<file>] -d <drive> [-l '<string>'] [-v [-v]]
```

'romwrite' kopiert eine als *UDOS*-Datei gespeicherte Imagedatei eines *RCBD-ROM*-Loaders auf ein Slice eines *RomWBW*-Datenträgers. Das Ziel kann eine mit der Option `rl=y` formatierte *RCBD-UDOS*-Diskette (*RCBD-ROM*-Loader-on-Disk) oder auch ein *RCBD-ROM*-Loader-only-Slice sein. Die Option `v` (Verbose) erzeugt zusätzliche Informationen in der Ausgabe. Die doppelte Angabe der Option `v` zeigt zusätzliche Information zur Imagedatei an.

Mit der Option `l` kann das Label im Boot Info Block der Imagedatei beim Kopieren in das Slice angepasst werden. Maximal 16 Zeichen werden übernommen. Soll das Label Leerzeichen enthalten, muss die zweite Form des Kommandos benutzt werden und das Label in einfache Anführungszeichen gesetzt werden.

Wir die `l` Option angegeben, aber kein Dateiname `file`, wird nur das Label eines bereits existierenden Boot Info Blocks im Slice neu gesetzt.

Die Imagedatei des *RCBD-ROM*-Loaders muss im *UDOS*-Dateityp `B` (Binary) mit einer Recordlänge von `0x100` (256 B) gespeichert sein. In Anlage O.5 ist ein Anwendungsbeispiel zu finden. Mögliche Fehlermeldungen beschreibt Anlage G.5

11.7 dam

```
Usage: dam <drv>
```

Das Dienstprogramm **'dam'** zeigt die Disk Allocation Map (DAM) der Diskette im Laufwerk `drv` an. `drv` muss eine gültige *UDOS*-Laufwerksnummer

sein. Liegt keine Diskette im Laufwerk, wird eine Fehlermeldung ausgegeben. Belegte und reservierte Sektoren werden durch ein `'A'` (allocated) dargestellt, freie Sektoren durch `'.'` und Sektoren, die auf Grund des Laufwerkstyps/Diskettenformats nicht existieren mit `'U'` (unusable).

Außerdem zeigt **'dam'** summarische Informationen zur Geometrie und zum Belegungsstatus der der Diskette an. Stimmen die in der DAM gespeicherten Belegungsdaten nicht mit der von **'dam'** selbst gezählten Anzahl freier/belegter Sektoren überein, wird eine inkonsistente Diskettenstatistik gemeldet.

11.8 y2kdate

```
Usage: y2kdate [yymmdd|-c|-r|-v]
```

'y2kdate' ist eine Anpassung des ursprünglichen **'date'** Dienstprogramms an die Jahrtausendwende. *UDOS* verwaltet die Zeitspempel für die Erstellung ('Date of creation') und der letzten Änderung ('Date of Modification') einer Datei in einem 6-stelligen Datenfeld in der Form `YYMMDD`. Dabei wird die zweistellige Jahreszahl implizit den Jahren 1900 bis 1999 zugeordnet.

'y2kdate' interpretiert die Datumsangabe bei der Anzeige des Systemdatums als 1970...1999 bei den Jahreszahlen 70...99 und als 2000...2069 bei den Jahreszahlen 00...69.

Das ist ein rein kosmetischer Vorgang, nur für die Anzeige des Systemdatums. Intern werden Dienstprogramme, die Datumsangaben vergleichen, trotzdem eine Datei mit dem Änderungsdatum '241115' als älter betrachten als eine Datei mit dem Änderungsdatum '890521' da $241115 < 890521$ ist.

Mit den Optionen `-c` oder `-r` kann das Systemdatum direkt aus der RTC gesetzt werden, sofern die Hardware eine solche Komponente beinhaltet.

```
-----------------------------------------------------------------------
rcbd:date                         ; Anzeige mit mit Original 'date'
Sonnabend, November 15, 1924      ; Datenfeld enthält '241115'
rcbd:y2kdate                      ; Anzeige mit mit 'y2kdate'
Freitag, November 15, 2024
rcbd:y2kdate 890521               ; Manuell ein (altes) Datum setzen
Sonntag, Mai 21, 1989
rcbd:y2kdate -c                   ; Datum aus RTC setzen (wenn vorh.)
Freitag, November 15, 2024
-----------------------------------------------------------------------
```

Die Originalversion des Dienstprogrammes **'date'** wurde aus Kompatibilitätsgründen auf der *RCBD*-Systemdiskette belassen.

11.9 set.date

```
Usage: set.date [-c|-r|-o|-D<yymmdd>|-v] [{-j|-y}<jahr>] \
                                [-m<monat>] [{-t|-d}<tag>]
```

Interaktives Setzen des Systemdatums. Die Datumselemente Jahr (`-j, -y`), Monat (`-m`) und Tag (`-t, -d`) können in der Kommandozeile angegeben werden, fehlende Elemente werden interaktiv abgefragt.

```
---------------------------------------------------------------------
rcbd:set.date
Tag?    (1...31) : 4
Monat?  (1...12) : 10
Jahr?   (0...99) : 24

Freitag, Oktober  4, 2024
rcbd:set.date -j24 -m11
Tag?    (1...31) : 15

Freitag, November 15, 2024
---------------------------------------------------------------------
```

Mit den Optionen `-c` oder `-r` (Datum aus RTC), `-o` (bisher eingestelltes Datum) oder `-Dyymmdd` (genau spezifiziertes Datum) kann auch ein Basisdatum gesetzt werden, von welchem mit den Jahr-, Monat- und Tag-Optionen einzelne Bestandteile selektiv verändert werden können.

```
---------------------------------------------------------------------
rcbd:set.date -c                        ; Datum aus RTC lesen

Sonnabend, November 16, 2024
rcbd:set.date -c -t10                   ; Datum aus RTC lesen und Tag
                                        ; modifizieren
Sonntag, November 10, 2024
---------------------------------------------------------------------
```

11.10 rtc

```
Usage: rtc [YYMMDDhhmmss|-v]
```

Das Dienstprogramm **'rtc'** ermöglicht das Auslesen und Anzeigen der kompletten Datums- und Zeitinformation aus der Echtzeituhr (sofern vorhanden). Außerdem ist darüber das Stellen der Echtzeituhr möglich. Ist keine von *RomWBW* unterstützte Echtzeituhr im System vorhanden, wird die Meldung `Invalid or inactive device` zurück gegeben.

Hinweis: 'rtc' ändert nicht das aktuelle Systemdatum des laufenden *UDOS*-Systems. Ein geändertes Datum in der Echtzeituhr wird entweder beim nächsten Start des *UDOS*-Systems als Systemdatum gesetzt oder es muss durch expliziertes Ausführen eines **'y2kdate'** oder **'set.date'** Kommandos mit der Option `-c` als Systemdatum übernommen werden.

11.11 do / ndo

Das Dienstprogramm **'do'** zur Ausführung von Command Files wurde nicht verändert. Da der Adressbereich, in den **'do'** geladen wird, sich nicht mit den Adressbereichen überschneiden darf, in die die im Command File enthaltenen Programme geladen werden, ist eine zweite Version von **do** auf der *RCBD*-Systemdiskette, die auf eine höhere Ladeadresse gelinkt wurde. Diese hat den (frei gewählten) Namen **'ndo'**. Mit diesen zwei Versionen von **do** sollten alle Fälle abgedeckt sein.

```
-----------------------------------------------------------------------
rcbd:cat f=l p=& *do do.obj
                  FILE RECORD RECORD FILE  START   DATE OF DATE OF
 FILENAME   DRIVE TYPE COUNT  LENGTH PROPS ADDRESS CREAT.  LAST MOD.

 do           0    P     1     0400  SF     7c00    791019  241108
 ndo          0    P     1     0400  SF     e000    870311  241108
 do.obj       0    B     6     0100                 861218  241108

    97 FILES EXAMINED
     3 FILES LISTED
    17 TOTAL SECTORS FOR LISTED FILES
-----------------------------------------------------------------------
```

Sollte der Anwender **do** auf eine weitere Adresse linken wollen, kann er die auf der Systemdiskette enthaltene Objektdatei **'do.obj'** gemäß seinen Wünschen linken. Dabei muss, abweichend von den Defaultwerten des Linkers die Stackgröße auf 256 B (`0x100`) gesetzt werden. Die Wahl der Recordlänge hat untergeordnete Bedeutung. Da **do** sich selbst aufrufen kann (ein Command File enthält den Aufruf eines anderen Command Files), muss **do** zur korrekten rekursiven Ausführung die Dateieigenschaft 'Force' (`F`) erhalten.

```
-----------------------------------------------------------------------
rcbd:ndo sys.link do 0c800 st=100 rl=400
PLINK 4.0
LINK COMPLETE
rcbd:set properties of 1/do to f
rcbd:cat f=l p=& *do
                  FILE RECORD RECORD FILE  START   DATE OF DATE OF
 FILENAME   DRIVE TYPE COUNT  LENGTH PROPS ADDRESS CREAT.  LAST MOD.

 do           0    P     1     0400  SF     7c00    791019  241108
 ndo          0    P     1     0400  SF     e000    870311  241108
 do           1    P     1     0400  F      c800    241118  241118

   100 FILES EXAMINED
     3 FILES LISTED
    15 TOTAL SECTORS FOR LISTED FILES
-----------------------------------------------------------------------
```

11.12 sys.link

```
Usage: do sys.link <object_file> <load_address> \
                          [<opt_arg1> [<opt_arg2> [<opt_arg3>]]]
```

'sys.link' ist kein Dienstprogramm, sondern ein Command File (Stapelverarbeitungs- oder Batchdatei). Ein Command File ist eine Textdatei, die eine Folge von *UDOS*-Dienstprogrammen mit einem einzigen Aufruf abarbeiten kann oder die wiederholte Ausführung von Dienstprogrammen mit einer Vielzahl von Kommandozeilenargumenten vereinfacht.

'sys.link' ist dazu gedacht, eigene Programme, die unter *UDOS* laufen sollen, gegen die globalen Systemadressen des Systemkerns zu linken. Das Command File kann mit der vorliegenden Parametrisierung immer benutzt werden, wenn das assemblierte Projekt nur aus einer einzigen Objekt-Datei besteht und die Startadresse (Entry Point) identisch mit der Ladeadresse ist. Für komplexere Projekte, in denen der Code auf mehrere Objektdateien verteilt ist, oder wo die Ladeadresse nicht mit dem Entry Point übereinstimmt, müssen individuelle Command Files erstellt werden, die das wiederholte Linken erleichtern.

'sys.link' legt mit der vorliegenden Parametrisierung das beim Linken entstehende Procedure File (ausführbare Programm) auf Laufwerk 1 ab, von wo es bei Bedarf auf das endgültige Ziellaufwerk kopiert werden kann. Daher muss bei Nutzung von **'sys.link'** in Laufwerk 1 eine beschreibbare Diskette liegen.

Als Beispiel das bekannte 'Hello world' Programm unter Nutzung einer globalen (public) Routine aus dem OS. Zuerst das Quellprogramm **'helloworld.s'**.

```
------------------------------------------------------------------
; Das Hello World Programm
; Aus OS:
        external PUTMSG         ; global in OS

        global   entry          ; Load- & start address

entry   ld      hl,hwmsg
        ld      c,(hl)
        ld      b,0             ; BC = Stringlength
        inc     hl
        call    PUTMSG          ; Call PUTMSG from OS
        ret                     ; Return to UDOS

hwmsg   deft    'Hello world'
        end
------------------------------------------------------------------
```

Die Erstellung des Procedure Files: Assemblieren und Linken. Der Assembler wird hier direkt aufgerufen, da der Aufruf sehr simpel ist. Der Linker muss die **'*.obj'**-Dateien des Systemkerns zusammen mit der Datei **'helloworld.obj'**, die beim Assemblieren entstanden ist, verbinden. Da das eine komplexe Kommandozeile ist, wurde das Linken durch Aufruf von **'sys.link'** ausgeführt.

Das Command File kann bis zu drei optionale Parameter an den Linker übergeben. In diesem Beispiel wurde die Recordlänge des Procedure Files spezifiziert sowie die benötigte Stackgröße. Die optionalen Parameter müssen nur angegeben werden, wenn der Linker von den Defaulteinstellungen abweichende Parameter benutzten soll.

```
---------------------------------------------------------------------
rcbd:asm helloworld
ASM 5.9 RCBD
PASS 1 COMPLETE
0 ASSEMBLY ERRORS
ASSEMBLY COMPLETE
rcbd:v                      ; Verbose um zu sehen, was ausgeführt wird
rcbd:do sys.link helloworld 4000 rl=200 st=100
plink -rcbdrom.dummy $=1000 -os.rcbd $=2100 -con.main -con.rcbd \
          $=2600 -ndos.dummy -last_module $=4000 helloworld \
          (nom e=4000 n=$NDOS:1/helloworld rl=200 st=100
PLINK 4.0
LINK COMPLETE
rcbd:cat f=l helloworld
                  FILE RECORD RECORD FILE  START   DATE OF DATE OF
 FILENAME   DRIVE TYPE COUNT  LENGTH PROPS ADDRESS CREAT.  LAST MOD.

 helloworld   1    P     1     0200         4000    241117  241117

   103 FILES EXAMINED
     1 FILES LISTED
     3 TOTAL SECTORS FOR LISTED FILES
rcbd:helloworld
Hello world
---------------------------------------------------------------------
```

11.13 sysinfo

```
Usage: sysinfo
```

'sysinfo' zeigt verschiedene Systemparameter wie *RomWBW*-Version, *RCBD-ROM*-Version, CPU-Parameter, Datum und Uhrzeit (wenn verfügbar) sowie die Austattung mit *RCBD*-geeigneten Massenspeichern an.

11.14 edit

'edit' ist bis auf eine winzige Änderung der originale zeilenorientierte Edi-

tor von *RIO* und *UDOS*. Die Änderung betrifft den Input Mode, in dem fortlaufend Textzeilen eingegeben werden können. Es folgt der Originaltext aus dem Manual:

```
In this mode, all text that is entered from the console is
inserted after the current line. Input mode is terminated
when a null line is entered (by typing just a carriage return).
```

Das Beenden des Input Modes durch alleinige Eingabe von `ENTER` bzw. `RETURN` am Anfang einer neuen Zeile hatte als Konsequenz, dass keine echten Leerzeilen im Text erzeugt werden konnten. Eine optisch leere Zeile im Text musste dadurch erzeugt werden, dass man am Anfang einer neuen Zeile ein einzelnes Leerzeichen eingeben musste (welches dann auch in der Datei gespeichert war), um im Input Mode zu verbleiben.

Das Verhalten wurde dahingehend geändert, dass im Input Mode das Drücken von `ENTER` bzw. `RETURN` direkt am Anfang einer neuen Zeile eine echte Leerzeile (die nur das Zeilenendezeichen beinhaltet) erzeugt. Zum Verlassen der Input Modes muss in der geänderten Editorversion in der neuen Zeile als erstes ein einzelner Punkt `'.'`, unmittelbar gefolgt von `ENTER` bzw. `RETURN` eingegeben werden.

11.15 vi

'vi' ist ein Clone des bekannten gleichnamigen Editors aus der Linuxwelt. Eigentlich handelt es sich um ein bildschirmorientiertes Frontend zum *UDOS*-Editor **'edit'**, dass mit den Tastaturbefehlen des **vi** bedient wird. **'vi'** erfordert, dass das benutzte Terminal ANSI Escapesequenzen verarbeiten kann.

Für dieses Programm wird es eine separate Dokumentation geben.

11.16 ttysize

```
Usage: ttysize [-d|-x]
       ttysize [{-r|-l|-z}<rows>] [{-c|-s}<columns>]
```

'ttysize' setzt zwei in **'rcbdrom.dummy.obj'** global vereinbarte 8 bit-Variable `TTYCOL` und `TTYLIN` im System-RAM, die die Bildschirmauflösung des verwendeten Konsolenterminals enthalten. Diese Variablen sind kein Bestandteil früherer *UDOS*-Systeme, sondern wurden im *RCBD* eingeführt, um zukünftige bildschirmorientierte Programme unter *UDOS* zu ermöglichen. Bisher werden diese Variablen nur im **vi**-Editor (11.15) benutzt.

Der Wert dieser Variablen muss mit der Auflösung des verwendeten Terminals oder den Einstellungen des verwendeten Terminalemulationsprogramms übereinstimmen und manuell gesetzt werden. Es gibt keinen Mechanismus, der es *RCBD-UDOS* erlaubt, die Auflösung des verwendeten Terminals selbst zu erkennen.

Initial wird während der Initialisierungsphase des *RCBD-ROM* eine Bildschirmauflösung von 80x24 Zeichen eingestellt, was die Standardauflösung bei vielen Hardwareterminals ist. Falls das verwendete Terminal eine andere Auflösung als 80x24 benutzt, oder bei Terminalemulationen, die eine frei wählbare Auflösung erlauben, kann nach dem *UDOS*-Start durch manuelle Ausführung von **'ttysize'** oder durch Einbinden von **'ttysize'** in die Datei **'os.init'** die tatsächlich benötigte Auflösung eingestellt werden.

'ttysize' ohne Optionen zeigt die aktuell eingestellte Auflösung an. Mit den Optionen `-r`, `-l` oder `-z` (Rows, Lines, Zeilen) kann die vertikale und mit `-c` oder `-s` (Columns, Spalten) die horizontale Auflösung eingestellt werden. Die beiden Werte können einzeln oder gemeinsam gesetzt werden. Es ist jeweils der Wertebereich von 16 bis 255 für die Parameter zulässig.

```
----------------------------------------------------------------------
rcbd:ttysize
TTY/Screen size 80x24
rcbd:ttysize -c137 -r34
TTY/Screen size 137x34
----------------------------------------------------------------------
```

Die Option `-d`, die allein anzugeben ist, stellt die Auflösung zurück auf den Defaultwert 80x24. Die Option `-x`, die allein anzugeben ist, stellt `TTYCOL` und `TTYLIN` auf Null, was für ein nutzendes Anwendungsprogramm das Zeichen sein sollte, dass keine Auflösungsinformation zur Verfügung steht.

```
----------------------------------------------------------------------
rcbd:ttysize -x
TTY/Screen size 0x0
rcbd:ttysize -d
TTY/Screen size 80x24
----------------------------------------------------------------------
```

11.17 xm

```
Usage: xm -s [-k] <send_file>
       xm -r [-a|-b] [-c] [-x] <receive_file>
```

Das Dienstprogramm **'xm'** zur Datenübertragung mit dem XMODEM-Protokoll wurde von der in *RomWBW* enthaltenen CP/M-Version abgeleitet, für die im Sourcezweig der *RomWBW*-Software der Quellcode enthalten ist. Die Übertragung erfolgt über die Konsolenschnittstelle, so dass der *RCBD* mit einem Terminalemulationsprogramm bedient werden muss, welches eine XMODEM Übertragung integriert hat (z. B. TeraTerm). **'xm'** arbeitet ausschließlich im Binärmode, d. h. es erfolgt keine Veränderung des Dateiinhalts, z. B. Anpassung von Zeilenendezeichen zwischen *RCBD* und dem Wirtsrechner.

11.17.1 XMODEM senden

'xm' überträgt Daten in 128 B Paketen und optional in 1 KiB Paketen, wenn die Option `-k` angegeben ist (Erweiterung XMODEM-1k). Die Datenübertragung wird bei 128 B Paketen durch eine Prüfsumme (Checksum) oder durch CRC (Erweiterung XMODEM-CRC) gesichert, den Mode fordert der Empfänger entsprechend seiner Implementierung an. Bei XMODEM-1k ist immer die Absicherung durch CRC vorgeschrieben.

Die Sendeblockgröße ist adaptiv. *UDOS*-Dateien sind stets ein Vielfaches von 256 B (Recordlength=0x100) lang. **'xm'** sendet, sofern die `-k` Option angegeben wurde, 1 KiB Pakete, bis der verbleibende Rest weniger als 1 KiB beträgt. Dieser Rest wird in 128 B Paketen übertragen, so dass die empfangene Datei auf dem Wirtsrechner exakt die gleiche Größe hat wie auf dem *UDOS*-Rechner.

```
----------------------------------------------------------------------
rcbd:xm -sk xm.s
XMODEM for RCBD 241216
(1k protocol selected)
File size: 194 records (25k)
To cancel: Ctrl-X, pause, Ctrl-X
----------------------------------------------------------------------
```

Die Datei **'xm.s'** wird in 1 KiB Paketen gesendet. Auf der Empfangsseite muss Speicherort und Dateiname im XMODEM-Programm des Empfängers angegeben werden, da das XMODEM-Protokoll keine Übertragung des Dateinamens vorsieht.

11.17.2 XMODEM empfangen (receive)

Beim Empfang von Dateien mit **'xm'** kann mit den Optionen `-a` oder `-b` angegeben werden, ob die Datei als *UDOS*-Dateityp `A` (ASCII) oder `B` (Binary) auf der *UDOS*-Diskette abgespeichert wird. `-b` ist default und kann auch weggelassen werden. Eine vorhandene Datei gleichen Namens wird überschrieben. Die Option `-c` fordert vom Sender explizit den Checksum-Mode an, die Option den `-x` den 128 B Mode. Inwieweit die sendende XMODEM-Seite diese Wünsche berücksichtigt, ist implementationsabhängig.

Dateien können auf dem Wirtsrechner (Windows- oder Linux-PC) eine beliebige Dateilänge haben. Durch die paketweise Übertragung muss das letzte Paket der Übertragung mit Füllzeichen (`0x1A`) auf 128 B oder 1 KiB aufgefüllt werden, wenn die Dateigröße der Sendedatei nicht zufällig ein exaktes Vielfaches der Paketgröße ist. Die abgespeicherte *UDOS*-Datei wird dadurch fast immer größer sein, als das Original auf dem Wirtsrechner.

```
----------------------------------------------------------------------
rcbd:xm -ra 1/asciidatei
XMODEM for RCBD 241216
File open - ready to receive
```

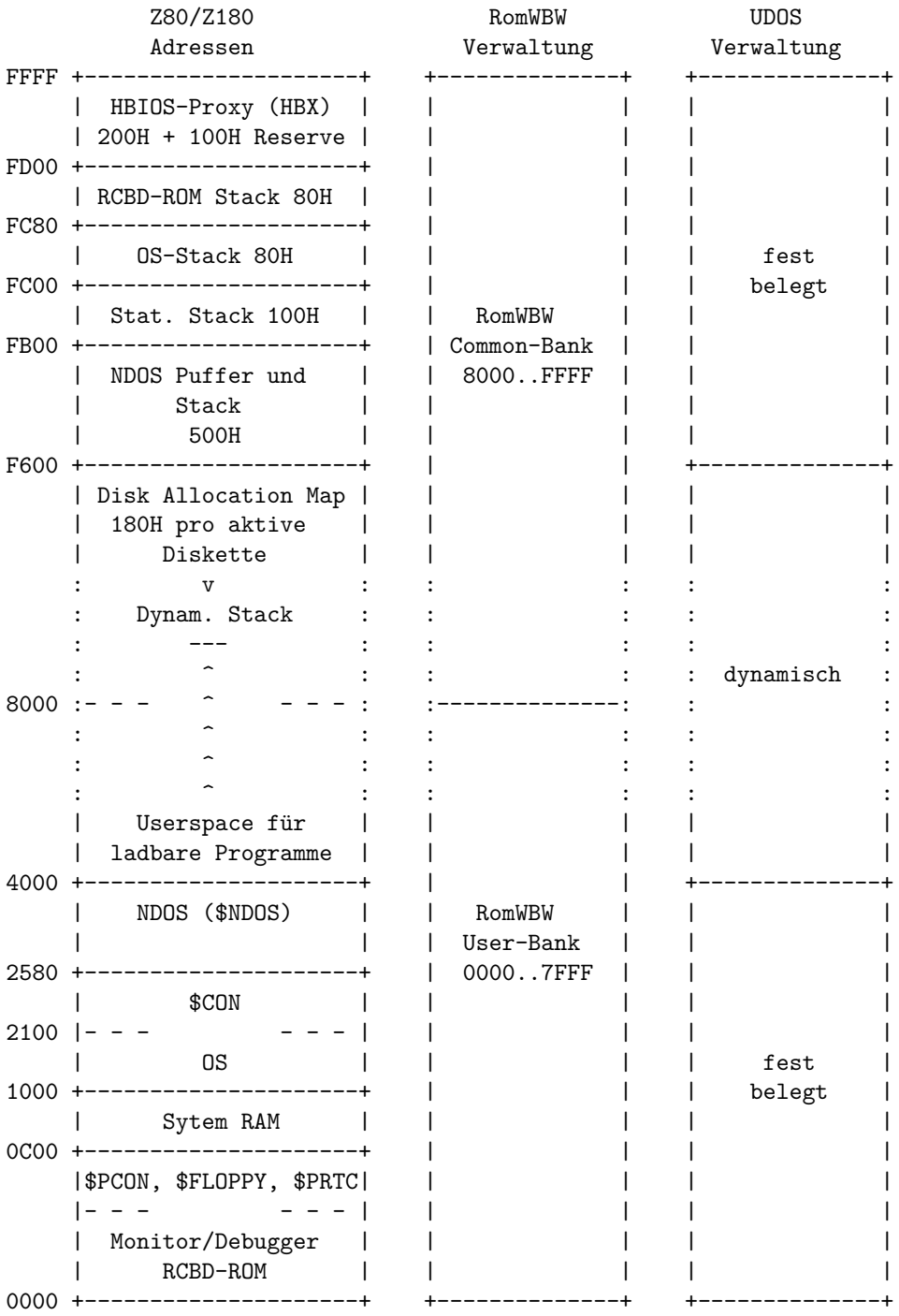
```
To cancel: Ctrl-X, pause, Ctrl-X
File size: 22 records (3k)
------------------------------------------------------------------------
```

Die vom Wirtsrechner gesendete Datei wird unter dem Namen **'asciidatei'** mit dem Dateityp `A` auf dem *UDOS*-Laufwerk 1 abgespeichert.

A Einstellungen Terminalprogramm (TeraTerm)

```
Übertragungsparameter
=====================================================================
Serial - COMx
Serial - 115200/8/n/1
                        (Speed entsprechend konkreter RomWBW Umgebung)
Serial - Flow Control - RTS/CTS
                    (oder 'Hardware', je nach Bezeichnung im Produkt)

Control-Set bzw. Terminalemulation
=====================================================================
ANSI Escape Sequenzen - nur für Bildschirmeditor vi notwendig

Sonst nur Carriage Return (0x0d), Line Feed (0x0a) und
Backspace (0x08) Behandlung notwendig.

Die RETURN bzw. ENTER Taste muss Carriage Return (0x0d) liefern.

In TeraTerm
=====================================================================
Serieller Port - Transmit delay: 0 msec/char, 0 msec/line
Terminal Einstellungen - Terminal-ID: VT100
Terminal Einstellungen - Lokales Echo: aus
Terminal Einstellungen - Neue Zeile: Übertrage CR, Absenden CR

Für optimale Nutzung der Cursortasten im vi-Editor muss im
Terminalemulationsprogramm (TeraTerm) eine Tastaturanpassungsdatei
geladen werden.

Die Flusssteuerung in RomWBW HBIOS erfolgt über Hardware.
Ein Download von Programmen oder Daten mit XMODEM oder Intel-HEX
vom Host-Rechner (PC) auf den RomWBW-Rechner kann vom 8-bit Rechner
nicht schnell genug verarbeitet werden. RomWBW signalisiert dem Host-
Rechner über die /RTS-Leitung die Bereitschaft zum Empfang von Daten.
```

B Speicherbelegung



Die Grundbelegung des Hauptspeichers, angezeigt mit dem Dienstprogramm **'display'**.

Die Ausgabe von **'display'** wurde hier redaktionell nachbearbeitet, um die fixe Speicherreservierung durch die Systemkomponenten darzustellen. Das `'A'` der belegten Speicherblöcke durch das gerade geladene Programm **'display'** wurden durch den Kleinbuchstaben `'d'` ersetzt. Die Belegung durch die Disk Allocation Map der Systemdiskette wurde durch den Kleinbuchstaben `'a'` ersetzt.

```
------------------------------------------------------------------------
rcbd:display
                    MEMORY ALLOCATION MAP

           0        400       800       C00
           |   |    |   |     |   |     |   |
0000       AAAAAAAA  AAAAAAAA  AAAAAAAA  AAAAAAAA
1000       AAAAAAAA  AAAAAAAA  AAAAAAAA  AAAAAAAA
2000       AAAAAAAA  AAAAAAAA  AAAAAAAA  AAAAAAAA
3000       AAAAAAAA  AAAAAAAA  AAAAAAAA  AAAAAAAA
4000       dddd....  ........  ........  ........
5000       ........  ........  ........  ........
6000       ........  ........  ........  ........
7000       ........  ........  ........  ........
8000       ........  ........  ........  ........
9000       ........  ........  ........  ........
a000       ........  ........  ........  ........
b000       ........  ........  ........  ........
c000       ........  ........  ........  ........
d000       ........  ........  ........  ........
e000       ........  ........  ........  ........
f000       ........  daaaAAAA  AAAAAAAA  AAAAAAAA

       "." => FREE    "A" => ALLOCATED
------------------------------------------------------------------------
```

C Datenträgergrößen

Kapazität	Größe RomWBW Partition	Anzahl Slices	Systemdisk in Slice	Eignung für RCBD/UDOS
8 MB		0	Medium nicht für RomWBW geeignet	
16 MB	8 MiB	1	0	bedingt brauchbar, nur eine UDOS-Disk möglich Einzige Möglichkeit: Systemdisk inclusive RCBD-ROM-Loader
32 MB	24 MiB	3	0	Systemdisk + Datendisk + sep. RCBD-ROM-Loader - oder - Systemdisk incl. RCBD- ROM-Loader + 2 Datend.
64 MB	56 MiB	7	0	brauchbar
128 MB	120 MiB	15	12	brauchbar
256 MB	240 MiB	30	12	brauchbar
512 MB	488 MiB	61	12	brauchbar
1GB	968 MiB	121	12	brauchbar
Maximal 2 GB RomWBW Partition (max. 256 Slices) bei Medien >2GB				

Die Angaben zur Eignung gehen davon aus, das auf dem Medium neben Datendisketten auch die zum Starten von *RCBD-UDOS* notwendigen Disketten bzw. Slices untergebracht sind. D.h. entweder eine Systemdiskette ohne *RCBD-ROM*-Loader und diesen in einem separaten Slice (*RCBD-ROM*-Loader-only-Slice), oder platzsparender eine Systemdiskette incl. *RCBD-ROM*-Loader (*RCBD-ROM*-Loader-on-Disk).

Datenträger haben in der Regel eine etwas geringere Kapazität als der aufgedruckte Handelswert. Auch schwankt die tatsächliche Kapazität zwischen Herstellern bzw. Modellen. Ich habe 128 MB CF-Karten unterschiedlicher Hersteller, die mit Kapazitäten zwischen 122 MiB und 125 MiB angezeigt werden.

Die *RomWBW*-Partitionierung für das hd1k-Layout erfordert 1 MiB Prefix für die Partitonstabelle und ein ganzahliges Vielfaches von 8 MiB für die *RomWBW*-Partition mit den Slices.

D Systemstart auf Z80-SBC

Hardware ist hier ein SC720[10] Z80 Single Board Computer (enthält bereits einen CF-Kartenadapter) und eine CTC Erweiterungskarte SC718[11]. Ein CH376-basierter USB-Controller[12] ergänzt die Massenspeicherausstattung.

Neben der RAM-Disk stecken zwei CF-Karten (32 MB, 128 MB) im System. Die angezeigte Kapazität der Karten ist 30 MiB bzw 125 MiB. Der 500 MB USB-Stick am CH376-USB-Controller wird mit 499 MiB angezeigt.

Ein Echtzeituhrmodul SC727[13] oder SC606[14] als Erweiterung ist geplant.

Die Konfiguration des *RomWBW* ROMs steht in Anlage F.

[10] https://smallcomputercentral.com/sc781-rcbus-z80-sbc-motherboard/
[11] https://smallcomputercentral.com/sc718-rcbus-z80-ctc-module/
[12] https://rc2014.co.uk/modules/ch375-usb-storage/
[13] https://smallcomputercentral.com/sc727-rcbus-rtc-module/
[14] https://smallcomputercentral.com/rcbus/sc600-series-selection/sc606-rcbus-rtc-module/

```
--------------------------------------------------------------------------
RomWBW HBIOS v3.5.1, 2025-07-10

RCBus [RCZ80_rcbd] Z80 @ 7.372MHz
0 MEM W/S, 1 I/O W/S, INT MODE 2, Z2 MMU
512KB ROM, 512KB RAM, HEAP=0x258B
ROM VERIFY: 00 00 00 00 PASS

CTC: IO=0x88
LCD: IO=0xDA NOT PRESENT
SIO0: IO=0x80 SIO MODE=115200,8,N,1
SIO1: IO=0x82 SIO MODE=115200,8,N,1
DSRTC: MODE=STD IO=0xC0 NOT PRESENT
MD: UNITS=2 ROMDISK=384KB RAMDISK=256KB
FD: MODE=RCWDC IO=0x50 NOT PRESENT
IDE: IO=0x10 MODE=RC
IDE0: ATA 8-BIT LBA BLOCKS=0x0000F500 SIZE=30MB
IDE1: ATA 8-BIT LBA BLOCKS=0x0003E800 SIZE=125MB
PPIDE: IO=0x20 PPI NOT PRESENT
SD: MODE=PIO IO=0x69 DEVICES=1
SD0: NO MEDIA
CH0: IO=0x3E TYPE=CH376 VER=0x04
CHUSB0: BLOCKS=0x00EB3FFF SIZE=499MB
CH1: IO=0x3C NOT PRESENT
FP: IO=0x00 NOT PRESENT

Unit        Device      Type              Capacity/Mode
----------  ----------  ----------------  --------------------
Char 0      SIO0:       RS-232            115200,8,N,1
Char 1      SIO1:       RS-232            115200,8,N,1
Disk 0      MD0:        RAM Disk          256KB,LBA
Disk 1      MD1:        ROM Disk          384KB,LBA
Disk 2      IDE0:       Hard Disk         30MB,LBA
Disk 3      IDE1:       Hard Disk         125MB,LBA
Disk 4      SD0:        SD Card           --
Disk 5      CHUSB0:     USB Drive         499MB,LBA

RCBus [RCZ80_rcbd] Boot Loader

Boot [H=Help]:
--------------------------------------------------------------------------
```

Der anschließende RCBD-Start sieht das System dann so:

```
------------------------------------------------------------------------
Boot [H=Help]: 2.2

Booting Disk Unit 2, Slice 2, Sector 0x00008800...

Volume "RCBD Z80 ROM" [0x0000-0x1FFF, entry @ 0x0000]...
Device 0 - RAM Disk
Device 2 - 03 Slice(s)*
Device 3 - 0f Slice(s)
Device 5 - 3e Slice(s)
No RTC found
UDOS for Z80 RomWBW HBIOS 250825 v2
>
------------------------------------------------------------------------
```

Device 2 ist das Default-Harddisk-Device, da das *RCBD-ROM* von diesem Device geladen wurde. Eine Echtzeituhr (RTC) wird nicht erkannt, da (noch) keine Hardware dafür vorhanden ist.

E Systemkonfiguration SC126 Z180-SBC

Die Systemkonfiguration des *RomWBW* ROMs für den SC126 Z180-Singleboardcomputer basiert auf der Standardkonfiguration **SCZ180_sc126_std**, die unverändert übernommen werden konnte.

```
------------------------------------------------------------------------
ROM Image File: SCZ180_sc126_std.rom
    Default CPU Speed 18.432 MHz
    Interrupts Mode 2
    System Timer Z180
    Serial Default 115200 Baud
    Memory Manager Z180
    ROM Size 512 KB
    RAM Size 512 KB

Supported Hardware
X FP:       LEDIO=0x0D
  FP:       SWIO=0x00
X DSRTC:    MODE=STD, IO=0x0C
  INTRTC:   ENABLED
X ASCI:     IO=0xC0, INTERRUPTS ENABLED
X ASCI:     IO=0xC1, INTERRUPTS ENABLED
  UART:     MODE=RC, IO=0xA0
  UART:     MODE=RC, IO=0xA8
  SIO       MODE=RC, IO=0x80, CHANNEL A, INTERRUPTS ENABLED
  SIO       MODE=RC, IO=0x80, CHANNEL B, INTERRUPTS ENABLED
  SIO       MODE=RC, IO=0x84, CHANNEL A, INTERRUPTS ENABLED
  SIO       MODE=RC, IO=0x84, CHANNEL B, INTERRUPTS ENABLED
X CH:       IO=0x3E
  CH:       IO=0x3C
X CHUSB:    IO=0x3E
  CHUSB:    IO=0x3C
X MD:       TYPE=RAM
X MD:       TYPE=ROM
  FD:       MODE=RCWDC, IO=0x50, DRIVE 0, TYPE=3.5" HD
  FD:       MODE=RCWDC, IO=0x50, DRIVE 1, TYPE=3.5" HD
X IDE:      MODE=RC, IO=0x10, MASTER
X IDE:      MODE=RC, IO=0x10, SLAVE
X PPIDE:    IO=0x20, MASTER
  PPIDE:    IO=0x20, SLAVE
X SD:       MODE=SC, IO=0x0C, UNITS=1

X = Als Hardware vorhanden
------------------------------------------------------------------------
```

F Systemkonfiguration SC720 Z80-SBC

Die Systemkonfiguration des *RomWBW*-ROMs für den SC720 Z80-Singleboardcomputer basiert auf der Standardkonfiguration **RCZ80_std**, die jedoch für den Einsatz im *RCBD* geändert werden musste. In der Standardkonfiguration wird im Interruptmode 1 gearbeitet, während für den *RCBD* der Interruptmode 2 notwendig ist.

```
------------------------------------------------------------------------
ROM Image File: RCZ80_rcbd.rom
    Default CPU Speed 7.372 MHz
    Interrupts Mode 2
    System Timer None
    Serial Default 115200 Baud
    Memory Manager Z2
    ROM Size 512 KB
    RAM Size 512 KB

Supported Hardware
X FP:       LEDIO=0x00
  FP:       SWIO=0x00
  DSRTC:    MODE=STD, IO=0xC0  (geplant)
  UART:     MODE=RC, IO=0xA0
  UART:     MODE=RC, IO=0xA8
X SIO       MODE=RC, IO=0x80, CHANNEL A, INTERRUPTS ENABLED
X SIO       MODE=RC, IO=0x80, CHANNEL B, INTERRUPTS ENABLED
  SIO       MODE=RC, IO=0x84, CHANNEL A, INTERRUPTS ENABLED
  SIO       MODE=RC, IO=0x84, CHANNEL B, INTERRUPTS ENABLED
  ACIA:     IO=0x80, INTERRUPTS ENABLED
X CH:       IO=0x3E
  CH:       IO=0x3C
X CHUSB:    IO=0x3E
  CHUSB:    IO=0x3C
X MD:       TYPE=RAM
X MD:       TYPE=ROM
  FD:       MODE=RCWDC, IO=0x50, DRIVE 0, TYPE=3.5" HD
  FD:       MODE=RCWDC, IO=0x50, DRIVE 1, TYPE=3.5" HD
X IDE:      MODE=RC, IO=0x10, MASTER
X IDE:      MODE=RC, IO=0x10, SLAVE
  PPIDE:    IO=0x20, MASTER
  PPIDE:    IO=0x20, SLAVE
X CTC:      IO=0x88

X = Als Hardware vorhanden
------------------------------------------------------------------------
```

Die Custom Configuration Datei für RCZ80 im Verzeichnis **<path>/Source/HBIOS/Config**.

```
---------------------------------------------------------------
; File:    Config/RCZ80_rcbd.asm
;==============================================================
; ROMWBW DEFAULT BUILD SETTINGS FOR RCBUS Z80 - Custom Configuration
;   - 2024-10-07 R.-P. Nerlich <early8bitz.de>
;   - Subject:   Modifications for RCBD
;   - Details:   Change interrupt mode to IM2
;                Increase RAM-Disk size from 256KiB to 352KiB
;==============================================================
;
#INCLUDE "Config/RCZ80_std.asm"
;
CPUOSC     .SET  7372800  ; CPU OSC FREQ IN MHZ
INTMODE    .SET  2        ; INTERRUPTS: 0=NONE, 1=MODE 1, 2=MODE 2,
;                         ;             3=MODE 3 (Z280)
; Optional in RomWBW 3.5.0 or later (not needed in RomWBW 3.4.x)
; Increase RAM-Disk size from 256KiB to 352KiB
; APP_BNKS   .SET  0        ; Default ist $FF which means Autoconfig
---------------------------------------------------------------
```

G System- und Fehlermeldungen

Es werden nur *RCBD*-spezifische Meldungen beschrieben. Meldungen aus der allgemeinen *RIO*- bzw. *UDOS*-Dokumentation werden hier nicht aufgeführt.

G.1 Lade- und Initialisierungsmeldungen

`Unsupported RomWBW Version, use 3.4.0 or higher`

Es wurde versucht, ein *RCBD-ROM*-Image auf einer zu alten *RomWBW*-Firmwareversion zu starten. *RCBD* wird nicht gestartet.

`Unsupported platform for Z80 RCBD-ROM detected`

Es wurde versucht, ein Z80-*RCBD-ROM*-Image auf einer Nicht-Z80-basierenden Hardware zu starten.
- oder -
Es wurde versucht, ein Z80-*RCBD-ROM*-Image auf einer Z80-basierenden Hardware zu starten, die aber nicht kompatibel zu *RCBD* ist.
RCBD wird nicht gestartet.

`Unsupported platform for Z180 RCBD-ROM detected`

Es wurde versucht, ein Z180-*RCBD-ROM*-Image auf einer Nicht-Z180-basierenden Hardware zu starten.
- oder -
Es wurde versucht, ein Z180-*RCBD-ROM*-Image auf einer Z180-basierenden Hardware zu starten, die aber nicht kompatibel zu *RCBD* ist.
RCBD wird nicht gestartet.

`No free CTC channel found`

Es wurde kein freier CTC-Kanal für den $FLOPPY-Treiber gefunden. Alle CTC-Kanäle sind als Taktgeber für eine SIO im *HBIOS* registriert.
RCBD wird nicht gestartet.

`Init Error XX`

Fehler beim Initialisieren ohne expliziten Fehlertext. `XX` kann sein
B0 - Fehler beim Lesen der *RomWBW*-Systeminformationen
B1 - Fehler beim Lesen der *RomWBW*-Bankinformationen
B2 - User-Bank-ID und Shadow-Bank-ID identisch
B3 - Ungültige Bankgröße (nicht 32KiB)
B4 - Fehler beim Lesen von Konsolenparametern

RCBD wird nicht gestartet.

`No avialable Disk Device`

RomWBW hat kein Diskdevice bereit gestellt, dass für *RCBD* geeignet ist. Es kann im *RCBD-ROM*-Monitor/Debugger gearbeitet werden, aber es ist kein *UDOS* startbar.

`Device 0 - RAM Disk`

Ein für *RCBD* nutzbares RAM-Disk-Device wurde gefunden.

`Device X - YY Slice(s)`

Ein für *RCBD* nutzbares Diskdevice wurde gefunden. Das Device `X` enthält `YY` Slices. Device- und Slicenummer sind hexadezimal angegeben.

`No RTC found`

In der Hardware wurde keine von *RomWBW* unterstützte Echtzeituhr (RTC) gefunden. Die *UDOS*-Variable **DATE** wird auf das Build-Datum des *RCBD-ROM* gesetzt. Das aktuelle Datum muss nach dem Start von UDOS manuell mit den Dienstprogrammen **'y2kdate'**, **'set.date'** oder **'date'** eingestellt werden.

`Date set from RTC`

In der Hardware wurde eine von *RomWBW* unterstützte Echtzeituhr (RTC) gefunden. Die *UDOS*-Variable **DATE** wird auf das von der RTC gelesene Datum gesetzt.

G.2 UDOS Startmeldungen

`c2 Disk-Error`

Keine Diskette im Laufwerk 0 beim Versuch, UDOS zu booten.

`No valid system disk`

Die Diskette im Laufwerk 0 ist keine *RCBD*-Systemdiskette.

`File not found`

Der *UDOS*-Bootloader meldet, wenn er die Dateien des Betriebssystemkerns **'OS'** oder **'NDOS'** nicht auf der Systemdikette findet.

```
------------------------------------------------------------------
>os
rcbdboot: Ok
osload:   File not found: OS
------------------------------------------------------------------
```

Ein fehlerfreies Laden des *UDOS*-Betriebssystems verläuft ohne Meldungen. Ausgaben können jedoch durch die Programme erzeugt werden, die zur automatischen Abarbeitung in der Datei **'os.init'** notiert sind.

G.3 Meldungen von PATCHOS

`Not running on a RCBD system`

Es wurde versucht **'PATCHOS'** auf einem anderen *UDOS*-System (z. B. P8000) zu starten.

`OS successful patched with HighStack Patch for RCBD`

Das Patch wurde erfolgreich auf das geladene **OS** angewendet.

`OS already patched with HighStack Patch for RCBD`

Es wurde versucht, das Patch ein zweites Mal anzuwenden, obwohl es schon geladen war.

`Invalid or not prepatched OS version loaded`

Beim Laden des Patches wurde festgestellt, dass die geladene **OS**-Version nicht die für den *RCBD* vorbereitete **OS**-Version ist. Das Patch wird nicht geladen.

`HighStack Patch for RCBD removed from OS`

Das Patch wurde vom geladenen **OS** erfolgreich entfernt.

`OS currently not patched with HighStack Patch for RCBD`

Es wurde versucht, das Patch zu entladen, obwohl es nicht geladen war.

G.4 Fehlermeldungen von 'md'

Das Dienstprogramm **'md'** gibt Fehlermeldungen beim Diskmapping als Text aus. Die residente Version im *RCBD-ROM* meldet die gleichen Fehler als Returncode `RC=XX`.

`Not running on a RCBD system`

Es wurde versucht **'md'** auf einem anderen *UDOS*-System (z. B. P8000) zu starten.

`Invalid UDOS drive number`

RCBD unterstützt 4 *UDOS*-Laufwerke (0...3). Diese Meldung wird ausgegeben, wenn beim Diskmapping eine Laufwerksnummer größer als 3 angegeben wird. In der *RCBD-ROM*-Version wird der Returncode `FF` zurück gegeben.

`Unsupported HBIOS device number`

RCBD kann die ersten zehn ermittelten Diskdevices von *RomWBW* nutzen (Devices 0...9). Vor *RCBD-ROM*- und **'md'**-Version 251125 wurden nur 8 *HBIOS*-Devices unterstützt. Diese Meldung wird ausgegeben, wenn beim Diskmapping eine Devicenummer größer als 9 angegeben wird. In der *RCBD-ROM*-Version wird der Returncode `FE` zurück gegeben.

`Unusable HBIOS device (no HD in hd1k format or no RAM disk)`

Die angegebene Devicenummer ist ein Diskdevice, welches keine *RomWBW*-Partiton enthält (nicht im 'Modern Layout' (hd1k) vorbereitet) und ist auch keine RAM-Disk. In der *RCBD-ROM*-Version wird der Returncode `FD` zurück gegeben.

`Invalid slice number for device`

Die im Mapping angegebene Slicenummer ist höher als die höchste Slicenummer, die das Device auf Grund seiner Kapazität beherbergen kann. In der *RCBD-ROM*-Version wird der Returncode `FC` zurück gegeben.

`Drive not empty (a different slice is already mapped)`

Das *UDOS*-Laufwerk, das gemappt werden soll, ist nicht leer. Es ist bereits eine Diskette aus einem anderen Slice gemappt. Das Mapping in dem Laufwerk ist vorher zu löschen, bevor ein anderes Slice gemappt werden kann. In der *RCBD-ROM*-Version wird der Returncode `FB` zurück gegeben.

`Slice already mapped to another drive`

Das im Mapping angegebene Slice ist bereits in ein anderes Laufwerk gemappt. Ein Slice kann nicht gleichzeitig in mehrere Laufwerke gemappt sein. In der *RCBD-ROM*-Version wird der Returncode `FA` zurück gegeben.

`Drive and device incompatible (FD in RAM drive or so)`

Es wird versucht, ein Slice auf einem Diskdevice in ein RAM-Disk-Laufwerk zu mappen. In der *RCBD-ROM*-Version wird der Returncode `F9` zurück gegeben.

`Default HD device not changed`

Das Default-Harddisk-Device kann nicht geändert werden, weil die angegebene Devicenummer kein für *RCBD* geeignetes Harddisk-Device ist oder kein Device mit dieser Nummer existiert.

G.5 Fehlermeldungen von 'romwrite'

`Not running on a RCBD system`

Es wurde versucht **'romwrite'** auf einem anderen *UDOS*-System (z. B. P8000) zu starten.

`Error XX accessing drive Y`

Beim Zugriff auf das Ziellaufwerk trat der *UDOS*-Fehler XX auf.

`<file>: Can't open file`

Die angegebene Imagedatei kann nicht gefunden oder aus einem anderen Grund nicht geöffnet werden.

`<file>: File type is not binary`

Die angegebene Imagedatei ist nicht im Dateityp Binary gespeichert.

`<file>: Record length is not 0x100`

Die angegebene Imagedatei ist nicht mit einer Recordlänge von `0x100` (256 B) gespeichert.

`<file>: Record count is not in range 6..38`

Die angegebene Imagedatei hat eine Recordanzahl, die nicht zur Struktur eines *RCBD-ROM*-Loader-Images passt.

`<file>: No ROM loader signature in file`

Die angegebene Imagedatei enthält keine gültige Signatur eines *RCBD-ROM*-Loaders.

G.6 Sonstige Meldungen

`Trap @ XXXX`

In einer Z180-basierenden *RCBD*-Umgebung hat die CPU versucht, einen ungültigen Opcode auf der Adresse `XXXX` auszuführen.

H Globale Adressen des RCBD-ROM-Monitors

Auflistung aller im *RCBD-ROM* (Datei **'rcbdrom.dummy.obj'**) global vereinbarten Systemeintrittspunkte und Variablen im System-RAM. Nicht kommentierte Einträge entsprechen den Globals des P8000-*UDOS*.

Bei einigen Globals weicht die Namensgebung in der P8000-*UDOS*-Dokumentation von den originalen RIO-Namen ab. Die Namen aus der RIO-Dokumentation wurden im *RCBD-ROM* zusätzlich global veröffentlicht und sind in der Liste entsprechend dokumentiert.

Im Robotron *UDOS*-1526 existiert eine weitere Version der Benennung einzelner Adressen, die im *RCBD-ROM* nicht global veröffentlicht ist, aber in der Liste dokumentiert wurde.

```
------------------------------------------------------------------
PLINK 4.0

LOAD MAP
MODULE      ORIGIN LENGTH
rcbdrom       0000   1000

LINKAGE INFO  1000   0000

GLOBAL     ADDRESS   MODULE
appBank       0EA6   rcbdrom        ; HBIOS ID Shadow Bank
BDEC          0BF7   rcbdrom        ; COMIN bei UDOS-1526
BRKFLG        0FC4   rcbdrom        ; DEBFL bei UDOS-1526
BRKRTN        0FC5   rcbdrom
BTOHE         0BDF   rcbdrom
CHRDEL        0FC3   rcbdrom
CONIBF        0D8A   rcbdrom        ; INBU  bei UDOS-1526
CONIVC        0ECE   rcbdrom
CONOBF        0D04   rcbdrom        ; OUTBU bei UDOS-1526
CONOVC        0EC3   rcbdrom
CURTO         0FCE   rcbdrom
CURTRK        0FD6   rcbdrom
DATE          0FA2   rcbdrom
DEBUG         0BFA   rcbdrom
DSEL          0EDD   rcbdrom
DSKSEL        0EDD   rcbdrom
DS_BUF        0E9D   rcbdrom
ERCODE        0FB4   rcbdrom
ETIME         0FAF   rcbdrom
EXTRET        0FB5   rcbdrom
FDCONF        0EEB   rcbdrom
FLOPPY        0BFD   rcbdrom
GETA          0BE8   rcbdrom        ; INA bei UDOS-1526
HSTACK        FC00   rcbdrom        ; Stat. Stack in Common Bank
INPTR         0FBB   rcbdrom        ; INBUAD bei UDOS-1526
```

```
LFCNT         0FC0   rcbdrom
LINDEL        0FC2   rcbdrom
MCSQ          0EF0   rcbdrom
MEMBOT        0FB9   rcbdrom
MEMTOP        0FB7   rcbdrom
NCRCE         0EE5   rcbdrom
NMIDSP        0F8C   rcbdrom
NSECTE        0EE7   rcbdrom
NTRKE         0EE9   rcbdrom
NULLCT        0FBF   rcbdrom
OSSTCK        FC80   rcbdrom        ; Stack von 'OS' in Common Bank
OUTAS         0BF1   rcbdrom        ; WRTXT bei UDOS-1526
OUTPTR        0FBD   rcbdrom
PCON          0BEE   rcbdrom
PDEBG1        0BF7   rcbdrom        ; RIO: BDEC  bei P8000-UDOS
PDEBUG        0BFA   rcbdrom        ; RIO: DEBUG bei P8000-UDOS
PGTCHR        0BF4   rcbdrom        ; RIO: SSIGN bei P8000-UDOS
PPRESS        0BE5   rcbdrom        ; RIO: PRESS bei P8000-UDOS
PPTMSG        0BF1   rcbdrom        ; RIO: OUTAS bei P8000-UDOS
PRESS         0BE5   rcbdrom
PROMPT        0FC1   rcbdrom
PRTC          0BDC   rcbdrom        ; Sprungvert.: $PRTC Treiber
PTRS          0E9F   rcbdrom        ; PTR   bei UDOS-1526
PTYBUF        0FCB   rcbdrom        ; RIO:
PTYFLG        0EF0   rcbdrom        ; RIO: STATQ bei P8000-UDOS
PTYGET        0BE8   rcbdrom        ; RIO: GETA  bei P8000-UDOS
PTYMSK        0FC9   rcbdrom        ; RIO:
PTYPUT        0BEB   rcbdrom        ; RIO: PUTA  bei P8000-UDOS
PUTA          0BEB   rcbdrom        ; OUTA  bei UDOS-1526
RCBDSIG       0004   rcbdrom        ; Kennung RCBD-ROM
RDCONF        0EED   rcbdrom        ; RAM-Disk Konfiguration
READY         0BE2   rcbdrom
RQSECT        0E9D   rcbdrom
RSTDSP        0F8A   rcbdrom
SSIGN         0BF4   rcbdrom        ; LDNECH bei UDOS-1526
STACK         FD00   rcbdrom        ; RCBD-ROM Stack in Common Bank
STATQ         0EF0   rcbdrom
TIME          0FA8   rcbdrom
TTYCOL        0FDF   rcbdrom        ; TTY-Size - Spaltenanzahl
TTYLIN        0FDE   rcbdrom        ; TTY-Size - Zeilenzahl
usrBank       0EA5   rcbdrom        ; HBIOS ID User Bank
USRSTK        0FA0   rcbdrom

PROGRAM rcbdrom -- 1000 BYTES
ENTRY: 0000
------------------------------------------------------------------------
```

I Laufwerkkstypen

Die Hardware des P8000 konnte mit verschiedenen Laufwerkstypen ausgestattet werden, was auch unterschiedliche Diskettenkapazitäten bedeutete. Um nicht unterschiedliche Systemdisketten verwalten zu müssen, konnte die konkrete Laufwerkskonfiguration nach dem Booten von UDOS mit dem Dienstprogramm **'setfd'** eingestellt werden. Lediglich das Laufwerk 0 (Bootlaufwerk) musste halbwegs zum Format der Systemdiskette passen. Laufwerke und Diskettenformate wurden über Typnummern identifiziert.

Im *RCBD* wurde das Konzept der Typnummern inclusive der vom P8000 bekannten Laufwerkstypen übernommen, obwohl nur ein einziger Laufwerkstyp und ein einziges Diskettenformat unterstützt wird (etwas Nostalgie muss sein). Zusätzlich wurden neue Typen eingeführt, um die RAM-Disk verwalten zu können.

```
----------------------------------------------------------------------
rcbd:setfd l
 (1) Undefined format
 (2) 5,25" DD, SS,  40 tracks, 16 sectors per 256 byte
 (3) 5,25" DD, SS,  40 tracks, 16 sectors per 256 byte, 80 track drv
 (4) 5,25" DD, SS,  80 tracks, 16 sectors per 256 byte
 (5) 5,25" DD, DS,  80 tracks, 32 sectors per 256 byte
 (6) 5,25" DD, DS,  40 tracks, 32 sectors per 256 byte
 (7) Undefined format
 (8) Undefined format
 (9) RAM Drive 256k 32 tracks, 32 sectors per 256 byte
 (A) RAM Drive 352k 44 tracks, 32 sectors per 256 byte
 (B) Undefined format
 (C) Undefined format
 (D) Undefined format
 (E) Undefined format
 (F) Undefined format
----------------------------------------------------------------------
```

Im *RCBD* `$FLOPPY`-Treiber werden nur die Formate 5, 9 und A unterstützt. Alle Laufwerke, die einen Typ ungleich 5, 9 oder A haben, werden beim Zugriff als 'Drive not ready' (Error C2) gemeldet.

J RAM-Disk-Kapazitäten

Die folgenden Angaben beziehen sich auf *RomWBW*-Systeme mit 512 KiB RAM-Ausstattung.

Bei *RomWBW*-Version 3.4.x ist die RAM-Disk-Kapazität immer 352 KiB. Die RAM-Disk kann unter *RCBD-UDOS* problemlos sowohl als Laufwerkstyp 9 (unter Verzicht eines Teils der Kapazität) als auch Typ A konfiguriert und formatiert werden.

Bei *RomWBW*-Version 3.5.x und der aktuellen Entwicklungsversion 3.6.0 ist die RAM-Disk-Kapazität in den mitgelieferten Standardkonfigurationen 256 KiB. Das erfordert seitens *RCBD-UDOS* unbedingt den Laufwerkstyp 9. Bei *RomWBW*-Version 3.5.x erfolgt im *HBIOS* allerdings keine Überwachung auf Zugriffe hinter die 256 KiB-Grenze, so dass eine versehentliche Konfiguration als Laufwerkstyp A beim Formatieren und Zugriff anfangs funktioniert, ab einem bestimmten Füllgrad aber zu Systemabstürzen führt. In *RomWBW*-Version 3.6.0 wird ein Zugriff oberhalb der Kapazitätsgrenze vom *HBIOS* abgefangen und als Fehler gemeldet.

Unter Verzicht auf die mit *RomWBW*-Version 3.5.0 eingeführten Applikationsbänke (Application Banks), die im *RCBD* nicht benötigt werden, kann die RAM-Disk mit der vollen Kapazität von 352 KiB genutzt werden. Dazu muss das HBIOS mit einer benutzerdefinierten Konfiguration (Custom Configuration) angepasst und ein individuelles *RomWBW*-ROM generiert werden.

Template einer Custom Configuration Datei im Verzeichnis **<path>/Source/HBIOS/Config**.

```
---------------------------------------------------------------------
; File:    Config/<platform>_rcbd.asm
;=====================================================================
; ROMWBW DEFAULT BUILD SETTINGS FOR <platform> - Custom Config.
;   - 2025-06-01 R.-P. Nerlich <early8bitz.de>
;   - Subject:   Modifications for RCBD
;   - Details:   Increase RAM-Disk size from 256KiB to 352KiB
;=====================================================================
;
#INCLUDE "Config/<platform>_std.asm"
;
; Other changed settings (if needed)
;
; Optional in RomWBW 3.5.0 or later (not needed in RomWBW 3.4.x)
; Increase RAM-Disk size from 256KiB to 352KiB
APP_BNKS   .SET  0        ; Default ist $FF which means Autoconfig
---------------------------------------------------------------------
```

`<platform>` ist zum Beispiel `RCZ80` oder `SCZ180_sc126`.

K Diskettenbelegung

UDOS verwaltet die Belegung der Disketten in der Disk Allocation Map (DAM), die in fest reservierten Sektoren jeder Diskette gespeichert wird. Dabei wird pro physischem Sektor ein Bit benutzt, welches entweder 0 für einen freien Sektor oder 1 für einen belegten Sektor ist. Insgesamt verwaltet die DAM 2560 Sektoren, basierend auf der maximalen Diskettengeometrie von 80 Spuren, 2 Köpfen und 16 Sektoren pro Spur und Seite. Das entspricht bei einer Sektorgröße von 256 byte einer Kapazität von 640 KiB.

Bei Datenträgern geringerer Kapazität (weniger Spuren u/o einseitigen Diskettenformaten) werden vom **'format'** Dienstprogramm die nicht existierenden Sektoren des jeweiligen Diskettenformates bereits beim Formatieren auf belegt setzt und das **'status'** Dienstprogramm zeigt die frei/belegt Statistik bezogen auf die tatsächlich nutzbare Sektoranzahl des jeweiligen Diskettenformats an.

Für die im *RCBD* benutzbaren Diskettenformate ergeben sich folgende Werte:

Disktyp	Media	Spuren	Köpfe * Sektoren	Sektoren total
5	Floppy	80	2 * 16	2560 (640KiB)
9	RAM	32	2 * 16	1024 (256KiB)
A	RAM	44	2 * 16	1408 (352KiB)

Die folgenden Tabellen zeigen die Disk Allocation Map von frisch formatierten Disketten. Der Buchstabe `A` markiert einen für das System reservierten Sektor, während ein Punkt einen freien, für das Dateisystem verfügbaren Sektor darstellt.

Alle Diskadressen sind 16 bit hexadezimal mit der Spurnummer im MSB und der Sektornummer im LSB. Die Zählung vom Spur- und Sektornummer beginnt jeweils bei Null und geht bis MAXTRK-1 bzw. MAXSEC-1.

K.1 Anwenderdiskette

Anwenderdiskette ohne Betriebssystem, nicht bootfähig. Die initiale Belegung ist identisch zu einer P8000 Anwenderdiskette.

Diese Diskette entsteht, wenn das **'format'** Dienstprogramm ohne die Optionen `s` und `rl=y` aufgerufen wird.

```
------------------------------------------------------------------
rcbd: format d=2 id='Anwenderdiskette'
------------------------------------------------------------------

------------------------------------------------------------------
Disk Allocation Map:
T 00 (0x00)  ........ ........ ........ ........  (0000 0000)
.....
T 21 (0x15)  ........ ........ ........ .......  (0000 0000)
T 22 (0x16)  AAA..AA. ..AA.... ........ .......  (E630 0000)
T 23 (0x17)  AA...... ........ ........ .......  (C000 0000)
T 24 (0x18)  ........ ........ ........ .......  (0000 0000)
Statistic: Sectors used=9; free=2551; Map count=9
------------------------------------------------------------------
```

```
-----------------------------------------------------------------
| Sektor(en)                  | Inhalt                          |
|-----------------------------+---------------------------------|
| 1600                        | Descriptor Record vom DIRECTORY |
| 1602                        | Pointerblock des DIRECTORY      |
| 1605, 160A, 1601, 1606, 160B | Datenrecords des DIRECTORY      |
| 1700, 1701                  | Disk Allocation Map (DAM)       |
-----------------------------------------------------------------
```

Die in den Tabellen bei `Statistic` angegebene Anzahl an freien Sektoren bezieht sich auf das Diskettenformat mit 80 Spuren. Bei den RAM-Disk-Formaten ist die Anzahl freier Sektoren entsprechend geringer. Der Wert von `used+free` ist bei einer konsistenten Disk Allocation Map immer die Gesamtzahl der nutzbaren Sektoren des jeweiligen Diskettenformats.

K.2 Anwenderdiskette mit ROM-Loader

Anwenderdiskette ohne Bootloader, nicht bootfähig. Zusätzlich ist der Bereich für den *RCBD-ROM*-Loader reserviert. Von dieser Diskette könnte das *RCBD-ROM* geladen, aber nicht das *UDOS*-Betriebssystem gebootet werden.

Diese Diskette entsteht, wenn das **'format'** Dienstprogramm mit der Option `rl=y` aufgerufen wird. Der *RCBD-ROM*-Loader-Code muss nach dem Formatieren separat in den reservierten Bereich kopiert werden.

```
------------------------------------------------------------------
rcbd: format d=2 id='User Disk + ROM Loader' rl=y
------------------------------------------------------------------

------------------------------------------------------------------
Disk Allocation Map:
T 00 (0x00)  ....AAAA AAAAAAAA AAAAAAAA AAAAAAAA  (0FFF FFFF)
T 01 (0x01)  AAAAAA.. ........ ........ ........  (FC00 0000)
.....
T 21 (0x15)  ........ ........ ........ ........  (0000 0000)
T 22 (0x16)  AAA..AA. ..AA.... ........ ........  (E630 0000)
T 23 (0x17)  AA...... ........ ........ ........  (C000 0000)
T 24 (0x18)  ........ ........ ........ ........  (0000 0000)
Statistic: Sectors used=43; free=2517; Map count=43
------------------------------------------------------------------

------------------------------------------------------------------
| Sektor(en)                  | Inhalt                          |
|-----------------------------+---------------------------------|
| Alle Sektoren einer Anwenderdiskette zzgl.                    |
| 0004, 0005                  | ROM-Loader Boot Info Block      |
| 0006...001F                 | ROM Code 0x0000...0x19FF        |
| 0100...0105                 | ROM Code 0x1A00...0x1FFF        |
------------------------------------------------------------------
```

K.3 Systemdiskette

Systemdiskette mit Bootloader, bootfähig. Die initiale Belegung ist identisch zu einer P8000 Systemdiskette. Von dieser Diskette kann das *UDOS*-Betriebssystem gebootet werden.

Diese Diskette entsteht, wenn das **'format'** Dienstprogramm mit der Option `s` aufgerufen wird. **'format'** kopiert den Bootloader und das GET/SAVE Paket von einer anderen Systemdiskette, die während des Formatierens verfügbar sein muss. Die regulären Systemdateien **'OS'** und **'NDOS'** sowie alle benötigten Dienstprogramme müssen manuell auf die frisch formatierte Systemdiskette kopiert werden.

```
------------------------------------------------------------------------
rcbd: format d=2 id='Systemdiskette' s
------------------------------------------------------------------------

------------------------------------------------------------------------
Disk Allocation Map:
T 00 (0x00)  A....... ........ ........ ........  (8000 0000)
.....
T 21 (0x15)  AAAAAAAA AAAAAAAA ........ ........  (FFFF 0000)
T 22 (0x16)  AAA..AAA AAAAAAAA ........ ........  (E7FF 0000)
T 23 (0x17)  AAAAAAAA AAAAAAAA ........ ........  (FFFF 0000)
T 24 (0x18)  ........ ........ ........ ........  (0000 0000)
Statistic: Sectors used=47; free=2513; Map count=47
------------------------------------------------------------------------

------------------------------------------------------------------------
| Sektor(en)   | Inhalt                                                |
|--------------+-------------------------------------------------------|
| Alle Sektoren einer Anwenderdiskette zzgl.                           |
| 0000         | Bootsektor (Primary Bootloader)                       |
| 1500...150F  | Rettungsbereich 1 für RAM während GET/SAVE            |
| 1607...1609  | OS Loader (Secondary Bootloader)                      |
| 160C, 160D   | Overlay GET/SAVE-Package (L=0200h)                    |
| 160E, 160F   | Rettungsbereich 2 für RAM während GET/SAVE            |
| 1702...170F  | Programmcode des GET/SAVE Packages (L=0E00h)          |
------------------------------------------------------------------------
```

K.4 Systemdiskette mit ROM-Loader

Systemdiskette mit Bootloader, bootfähig. Zusätzlich ist der Bereich für den *RCBD-ROM*-Loader reserviert. Von dieser Diskette könnte das *RCBD-ROM* geladen und das *UDOS*-Betriebssystem gebootet werden.

Diese Diskette entsteht, wenn das **'format'** Dienstprogramm mit beiden Optionen `s` und `rl=y` aufgerufen wird. **'format'** kopiert den Bootloader und das GET/SAVE Paket von einer anderen Systemdiskette, die während des Formatierens verfügbar sein muss. Die regulären Systemdateien **'OS'** und **'NDOS'** sowie alle benötigten Dienstprogramme müssen manuell auf die frisch formatierte Systemdiskette kopiert werden. Der *RCBD-ROM*-Loader-Code muss nach dem Formatieren separat in den reservierten Bereich kopiert werden.

```
----------------------------------------------------------------------
rcbd: format d=2 id='Sys-Disk + ROM Loader' s rl=y
----------------------------------------------------------------------

----------------------------------------------------------------------
Disk Allocation Map:
T 00 (0x00)  A...AAAA AAAAAAAA AAAAAAAA AAAAAAAA  (8FFF FFFF)
T 01 (0x01)  AAAAAA.. ........ ........ ........  (FC00 0000)
.....
T 21 (0x15)  AAAAAAAA AAAAAAAA ........ ........  (FFFF 0000)
T 22 (0x16)  AAA..AAA AAAAAAAA ........ ........  (E7FF 0000)
T 23 (0x17)  AAAAAAAA AAAAAAAA ........ ........  (FFFF 0000)
T 24 (0x18)  ........ ........ ........ ........  (0000 0000)
Statistic: Sectors used=81; free=2479; Map count=81
----------------------------------------------------------------------

----------------------------------------------------------------------
| Sektor(en)                  | Inhalt                             |
|-----------------------------+------------------------------------|
| Alle Sektoren einer Systemdiskette zzgl.                         |
| 0004, 0005                  | ROM-Loader Boot Info Block         |
| 0006...001F                 | ROM Code 0x0000...0x19FF           |
| 0100...0105                 | ROM Code 0x1A00...0x1FFF           |
----------------------------------------------------------------------
```

L RCBD-ROM-Monitor

L.1 Allgemeine Hinweise

Im *RCBD-ROM*-Monitor/Debugger sind alle Adress- und Zahlenangaben hexadezimal ohne Pre- oder Postfix. Die Schreibweise der Hexadezimalziffern (groß oder klein) ist nicht relevant, bei Ausgaben werden Kleinbuchstaben verwendet. Die Schreibweise der Kommandos ist ebenfalls case insensitiv.

Die Eingabe von hexadezimalen Zahlen wird im Kontext des jeweiligen Monitorkommandos als 8-Bit- oder als 16-Bit-Zahl verwendet. Führungsnullen können weggelassen werden. Werden mehr Stellen als erforderlich eingegeben, sind die nur die letzten zwei Stellen (bei 8-Bit-Werten) oder die letzten vier Stellen (bei 16-Bit-Werten) relevant.

Für 8-Bit-Werte, die gleichzeitig darstellbare 7-Bit-ASCII-Zeichen sind, kann an Stelle der hexadezimalen Eingabe das entsprechende ASCII-Zeichen mit vorangestellten Apostroph verwendet werden. So ist z. B. `'E` identisch mit dem 8-Bit-Wert `0x45`.

Da *RCBD* mit Interruptunterstützung arbeitet, sind folgende Hinweise bei der Arbeit mit dem *RCBD-ROM*-Monitor/Debugger zu beachten:

- Ein Anwenderprogramm darf das `I`-Register (Interruptvector-Register) und den Interruptmode `IM 2` nicht verändern.
- Ein Anwenderprogramm darf den maskierbaren Interrupt nicht dauerhaft disablen. Notwendige `DI...EI`-Abschnitte sind möglichst kurz zu halten.
- Beim Einzelschrittbetrieb sollte das Anwenderprogramm keinen `DI`-Befehl (Disable Interrupt) ausführen. In einem `DI...EI`-Abschnitt ist kein Einzelschrittbetrieb möglich. `DI...EI`-Abschnitte inclusive des dem `EI` folgenden Befehls werden im Einzelschrittbetrieb in einem Stück durchlaufen. Erst danach meldet sich der *RCBD-ROM*-Monitor/Debugger zurück.
- Beim Erreichen eines gesetzten Breakpoints geht die Kontrolle an den *RCBD-ROM*-Monitor/Debugger zurück und die CPU befindet sich im `EI`-Zustand (Interrupt enabled). Das gilt auch, wenn sich der Breakpoint innerhalb eines `DI...EI`-Abschnitts (Interrupt disabled) des Anwenderprogramms befindet.
- Der Stackpointer muss in allen Programmsituationen auf eine Adresse oberhalb von `0x8000` verweisen.

L.2 Kommandoreferenz

```
>b adresse [anzahl]
```

B[reak]: Bei Angabe von adresse wird ein evtl. gesetzter Breakpoint gelöscht und ein neuer auf die angegebene Adresse gesetzt. Die optionale Angabe anzahl setzt einen Zähler, so dass erst beim anzahl-ten Erreichen der Breakpointadresse in den Monitor zurück gekehrt wird. Fehlt diese Angabe gilt anzahl=1.

adresse=0 löscht einen gesetzten Breakpoint ohne einen neuen zu setzen. Auf die Adresse 0x0000 kann daher niemals ein Breakpoint gesetzt werden.

Ein Breakpoint muss immer auf die Adresse des ersten Bytes (erster Opcode) eines Maschinenbefehls gesetzt werden. Ein gesetzter Breakpoint wird nur aktiviert, wenn das zu debuggende Programm mit dem Go-Kommando g [startadr] gestartet wird.

```
>b
aaaa cc + ['q' | adresse [anzahl]]
```

Ohne Argument wird die aktuelle Breakpointadresse aaaa und der Breakzähler cc angezeigt und eine Aufforderung zur Eingabe einer neuen Breakpointadresse ausgegeben ('+'-Prompt). Soll die aktuelle Breakpointadresse nicht geändert werden, kann die Eingabeaufforderung mit ENTER oder q quittiert werden.

```
>c beginnadr_1 beginnadr_2 anzahl
```

C[ompare]: Vergleicht zwei Speicherblöcke Byte für Byte. Die Blöcke werden entsprechend der angeforderten Blocklänge anzahl Byte für Byte verglichen und die Differenzen ausgegeben.

```
>d beginnadr anzahl
```

D[isplay]: Ein Speicherblock beginnend bei beginnadr und der Größe von anzahl Bytes wird hexadezimal angezeigt. Zusätzlich wird das 7-Bit-ASCII-Äquivalent zu jedem Byte angezeigt, sofern es sich um ein darstellbares Zeichen handelt.

```
>d beginnadr
aaaa bb + ['q' | nn*]
```

Ohne Angabe einer Byteanzahl geht D[isplay] in den Speicheränderungsmode. Die angegebene Beginnadresse aaaa und das dort gespeicherte Byte bb werden angezeigt und eine Aufforderung zur Eingabe eines neuen Speicherinhaltes ausgegeben ('+'-Prompt). Soll das Byte auf

der angezeigten Adresse nicht geändert werden, kann die Eingabeaufforderung mit `ENTER` bestätigt werden, wodurch das Kommando ohne Änderung zur folgenden Speicheradresse weiter geht. Die Eingabe eines neuen Wertes `nn` überschreibt die aktuelle Speicheradresse. Im Rahmen der Eingabepuffergröße können hinter dem Pluszeichen auch mehrere Werte (Bytefolgen) notiert werden, die dann auf aufeinanderfolgende Speicheradressen geschrieben werden.

Der Speicheränderungsmode wird verlassen, indem am `'+'`-Prompt oder als letzter Wert einer Bytefolge ein `q` eingegeben wird.

```
------------------------------------------------------------------------
>d 4000 10
4000 33 cc 55 aa 33 cc 55 aa 33 cc 55 aa 33 cc 55 aa  >3.U.3.U.3.U.3.U.<
>d 4000
4000 33 +22                          ; Neuer Wert in Hex
4001 cc +'H 61 'l 'l 'o 22           ; Folge neuer Werte in ASCII und Hex
4007 aa +0                           ; Neuer Wert in Hex
4008 33 +                            ; ENTER = Keine Änderung
4009 cc +q                           ; 'q' = Änderungsmode verlassen
>d 4000 10
4000 22 48 61 6c 6c 6f 22 00 33 cc 55 aa 33 cc 55 aa  >"Hallo".3.U.3.U.<
------------------------------------------------------------------------
```

`>f beginnadr anzahl wert+`

`F[ill]`: Füllt den Speicher ab `beginnadr` mit dem Byte `wert`, bis `anzahl` aufeinanderfolgender Speicheradressen beschrieben ist. `wert` kan ein einzelnes Byte sein oder eine Bytefolge, die so lange wiederholend in den Speicher geschrieben wird, bis `anzahl` erreicht ist.

`>g [startadr]`

`G[oto]`: Das Programm auf Adresse `startadr` wird als Unterprogramm aufgerufen. Die Rückkehradresse zum *RCBD-ROM*-Monitor/Debugger liegt auf dem Stack, so dass das Unterprogramm mit einem `RET`-Befehl ordnungsgemäß beendet werden kann. Das Unterprogramm arbeitet im Kontext einer virtuellen CPU, deren Registerinhalte über das `R[egister]`-Kommando gesetzt u/o angezeigt werden können. Das Stackpointerregister `SP` der virtuellen CPU muss auf einer Speicheradresse in der oberen RAM-Bank zeigen. Wird keine Startadresse `startadr` angegeben, wird das Programm auf der Adresse gestartet, auf die der Programmzähler `PC` der virtuellen CPU aktuell zeigt.

Sofern ein Breakpoint gesetzt ist, wird dieser vor dem Programmstart aktiviert.

Es sollte darauf geachtet werden, dass im gestarteten Programm der Stack bei maximaler Nutzung nicht unter `0x8000` sinkt.

`>j startadr [ arg]*`

`J[ump]`: Das Programm auf Adresse `startadr` wird als Unterprogramm aufgerufen. Die Rückkehradresse zum *RCBD-ROM*-Monitor/Debugger liegt auf dem Stack, so dass das Unterprogramm mit einem `RET`-Befehl ordnungsgemäß beendet werden kann. Das Unterprogramm arbeitet im Kontext des *RCBD-ROM*-Monitor/Debuggers und benutzt dessen Stack. Registerpaar `DE` zeigt im Kommandozeilenpuffer auf das Trennzeichen hinter `startadr` (Command-String-Pointer), so dass dem Unterprogramm optional eine Argumentenliste über die Kommandozeile übergeben werden kann.

Es sollte darauf geachtet werden, dass das aufgerufene Programm den Stack nicht tiefer als 120 B in Anspruch nimmt.

`>m beginnadr_ziel beginnadr_quelle anzahl`

`M[ove]`: Verschiebt den Speicherblock der Größe `anzahl` ab `beginnadr_quelle` in den Speicherbereich, der bei Adresse `beginnadr_ziel` beginnt. Quell- und Zielspeicherbereich können sich überlappen.

`>md [drv]`

`M[ap]D[isk]` oder `M[ount]D[isk]`: Ohne Argument wird das aktuelle Diskmapping angezeigt. Die alleinige Angabe einer *UDOS*-Laufwerksnummer `drv` löscht ein vorhandenes Mapping.

`>md drv[|0x80] ddss`

Bei zwei Argumenten ist das erste Argument die Laufwerksnummer. Device- und Slicenummer sind als 16-Bit-Zahl `ddss` zusammengefasst, wobei das MSB die Devicenummer enthält und das LSB die Slicenummer. Daher muss die Slicenummer `ss` immer zweistellig angegeben werden, um die Abgrenzung zur Devicenummer `dd` zu ermöglichen. Mögliche Fehlermeldungen des Kommandos stehen in Anlage G.4.

Standardmäßig wird eine Diskette schreibgeschützt gemappt. Der Schreibschutz einer Diskette wird aufgehoben, indem beim Mappen die Laufwerksnummer mit gesetztem Bit 7 (ODER-Verknüpfung der Laufwerksnummer mit `0x80`) notiert wird.

`>n [anzahl]`

`N[ext]` führt den Befehl, auf den der Programmzähler `PC` der virtuellen CPU zeigt, im Kontext der virtuellen CPU im Einzelschrittbetrieb aus und zeigt danach den sich ergebenden Zustand wie beim `R[egister]`-Kommando an. Ist `anzahl` mit angegeben, wird die entsprechende Anzahl Befehle Schritt für Schritt ausgeführt, ohne die Angabe gilt `anzahl=1`.

Nach der Ausführung eines oder meherer Befehle im Einzelschrittbetrieb ändert sich das Prompt des Monitors in `'*'`. An diesem Prompt wirkt das alleinige Drücken der `ENTER`-Taste wie ein `N 1` Kommando, d.h. man kann durch fortlaufendes Drücken von `ENTER` ohne weitere Kommandos tippen zu müssen, durch ein Programm steppen. Jedes andere reguläre Monitorkommando kann am `'*'`-Prompt ebenfalls eingegeben werden, danach ist der Monitor wieder am normalen `'>'`-Prompt

`>os`

`OS:` Es wird von der Diskette im Laufwerk 0 der primäre Bootloader geladen und, wenn erfolgreich validiert, der Bootvorgang von UDOS eingeleitet. Eine fehlende Diskette in Laufwerk 0 oder eine Nicht-*RCBD*-Systemdiskette mit gemäß Anlage G.2 mit einem Fehler quittiert.

`>p portadr [wert]* ['q']`

`P[ort]` dient zum Lesen u/o Schreiben von I/O-Ports. Ohne Angabe eines Wertes als Argument wird das I/O-Port `portadr` mit einem `IN A,(C)` Befehl eingelesen, der gelesene Wert angezeigt und eine Aufforderung zur Eingabe eines neuen Wertes ausgegeben (`'+'`-Prompt). Wird an Stelle eines neuen Wertes `q` oder `ENTER` eingegeben, ist das Kommando beendet. Werden ein oder mehrere Werte am `'+'`-Prompt eingegeben, so werden diese nacheinander mit einem `OUT (C),A` Befehl an `portadr` gesendet.

Bei Angabe eines Wertes oder einer Werteliste in der Kommandozeile werden diese nacheinander mit einem `OUT (C),A` Befehl an `portadr` gesendet, ohne vorher einen `IN`-Befehl auszuführen.

Die `portadr` wird als 16-Bit-Wert behandelt. Der niederwertige Teil der Portadresse wird während des `IN`- oder `OUT`-Befehls auf den unteren Adressbus gelegt, der höherwertige Teil auf den oberen Adressbus.

```
-----------------------------------------------------------------
>p 38                   ; Port 0x38 lesen
78 +                    ; und nichts schreiben
>p 38                   ; Port 0x38 lesen
78 +00 40               ; und Wert(e) schreiben
>p 38 00 40             ; Wert(e) auf Port 0x38 schreiben ohne lesen
-----------------------------------------------------------------
```

`>q`

`Q[uit]:` Falls der *RCBD-ROM*-Monitor/Debugger vom **OS** oder einem Anwendungsprogramm als Unterprogramm aufgerufen wurde, kehrt man mit `q` zur aufrufenden Programmebene zurück. Andernfalls wird `q` ignoriert.

`>qx`

Mit `qx` erfolgt eine Rückkehr zum *RomWBW*-Bootprompt (Warmstart). Eine direkte Rückkehr in den *RCBD-ROM*-Monitor/Debugger ist von dort nicht möglich. Es muss ein regulärer Neustart des *RCBD-ROM* gemäß den Punkten 7.2 oder 7.2.2 durchgeführt werden.

```
>r
```

`R[egister]` ohne Argument zeigt alle Register der virtuellen CPU an. In der oberen Zeile werden der Hauptregistersatz sowie `IX` und `PC` angezeigt. In der unteren Zeile werden der Alternativregistersatz sowie `IY` und `SP` angezeigt. Die Flagregister `F` und `F'` sind bitweise aufgeschlüsselt. Zu den Registern `PC` und `SP` sind zusätzlich die nächsten vier Byte aus dem Hauptspeicher angegeben, auf die diese Register zeigen. In der Spalte für das `I`-Register zeigt die obere Zeile das `I`-Register selbst an, die untere Zeile das Interrupt Enable Flip-Flop `IFF`.

```
>r register
 reg rr[rr] + ['q' | nn[nn]]
```

Wird als Argument ein Registername angegeben, geht das `R[egister]`-Kommando in den Registeränderungsmode über. Der angegebene Registername `reg` und der darin enthaltene Inhalt `rr` (`rrrr` bei 16-Bit-Registern) werden angezeigt und eine Aufforderung zur Eingabe eines neuen Registerinhaltes ausgegeben (`'+'`-Prompt). Soll der Inhalt des angezeigten Registers nicht geändert werden, kann die Eingabeaufforderung mit `ENTER` bestätigt werden, wodurch das Kommando ohne Änderung zum folgenden Register wechselt. Die Eingabe eines neuen Wertes `nn` bzw. `nnnn` überschreibt den aktuellen Registerinhalt.

Der Registeränderungsmode wird verlassen, indem am `'+'`-Prompt oder hinter einem neuen Inhalt ein `q` eingegeben wird.

```
-----------------------------------------------------------------------
>r
sz.h.pnc a  b  c  d  e  h  l  i  ixiy pcsp  mem
00000000 00 00 00 00 00 00 00 ff 0000 0000 f3 c3 de 06
00000000 00 00 00 00 00 00 00 01 0000 fcd0 ff ff ff ff
>r d
d  00 +01                          ; Register D ändern
e  00 +80                          ; Register E ändern
h  00 +q                           ; 'q' = Änderungsmode verlassen
>r f'
f' 00 +01                          ; Register F' ändern
a' 00 +55 q                        ; Register A' ändern und quit
>r pc
pc 0000 +4000                      ; Register PC ändern
sp fcd0 +q
>r
```

```
sz.h.pnc a  b  c  d  e  h  l  i  ixiy pcsp  mem
00000000 00 00 00 01 80 00 00 ff 0000 4000 3e 25 2a bb
00000001 55 00 00 00 00 00 00 01 0000 fcd0 ff ff ff ff
-----------------------------------------------------------------
```

```
>x startadr [ arg]*
```

`X[ecute]:` Ähnlich wie `J[ump]`, aber die Übergabe der optionalen Argumente aus der Kommandozeile ist erweitert und entspricht den Konventionen beim Aufruf externer Programme durch das **OS**. Das Programm auf Adresse `startadr` wird als Unterprogramm aufgerufen, bekommt aber anders als `J[ump]` einen eigenen Stack mit auf den Weg. Die Rückkehradresse zum *RCBD-ROM*-Monitor/Debugger liegt auf dem Stack, so dass das Unterprogramm mit einem `RET`-Befehl ordnungsgemäß beendet werden kann. Registerpaar `DE` zeigt im Kommandozeilenpuffer auf das Trennzeichen hinter `startadr` (Command-String-Pointer), so dass dem Unterprogramm optional eine Argumentenliste über die Kommandozeile übergeben werden kann. Zusätzlich liegt der Command-String-Pointer hinter der Rückkehradresse auf dem Stack und in der globalen Variable `INPTR` (Adresse `0x0FBB`).

Es sollte darauf geachtet werden, dass das aufgerufene Programm den Stack nicht tiefer als 72 B in Anspruch nimmt.

L.3 Manuelle Programmunterbrechung (Break)

Der Programmablauf, speziell beim Debuggen von Programmen, kann durch Auslösen eines NMI-Signals unterbrochen werden. Der *RCBD-ROM*-Monitor/Debugger sichert den aktuellen Zustand (Register) des unterbrochenen Programms und zeigt die gesicherten Register an. Es kann danach wie auch beim Erreichen eines Softwarebreakpoints (Debuggerkommando `b`) weiter verfahren werden.

Die NMI-auslösende Taste oder Schaltung muss prellfrei einen Low-Impuls liefern. Falls schaltungstechnisch mehrere NMI-Quellen angeschlossen werden sollen, müssen diese als Wired-AND-Schaltung aufgebaut sein. D. h. die einzelnen NMI-Quellen müssen mit Open-Kollektor- oder Open-Drain-Stufen alternativ den Low-Pegel am /NMI-Anschluss der CPU erzeugen. Ein Pullup-Widerstand nach High (Vcc) sorgt für High-Pegel am /NMI-Anschluss, wenn keine NMI-Quelle aktiv ist.

Der NMI führt nicht zur Programmunterbrechung, wenn *RCBD* gerade nicht im nativen *UDOS*-Adressraum arbeitet, d. h. wenn Kode im *HBIOS* oder in der Shadow-Bank des *RCBD* ausführt wird. Ebenso wird das Programm nicht unterbrochen, wenn im nativen *UDOS*-Adressraum Kode im *HBIOS*-Proxy (Adressen `0xFD00-0xFFFF`) oder im *RCBD-ROM*-Monitor-Bereich `0x0000-0x0BFF` ausführt wird.

Die korrekte NMI-Unterbrechungsbehandlung ist erst ab *RCBD-ROM*-Version 251008 gewährleistet.

M Aufruf des Treibers $PRTC

`$PRTC` ist als *UDOS*-Treiber geschrieben. Der Aufruf erfolgt über den standardisierten I/O-Vektor, dessen Anfangsadresse im Register `IY` zu übergeben ist. Die Aufrufadresse ist die Adresse `0x0BDC` im Sprungverteiler am Ende des *RCBD-ROM*.

```
I/O-Vektor für den Aufruf des Treibers
IY+0      0x00  Unit, unbenutzt
IY+1      0xrr  rr = Request Code
IY+2,3  0xaaaa  aaaa = Puffer Adresse für Datenfeld
IY+4,5  0x0006  Datenlänge, wird momentan ignoriert
IY+6,7  0x0000  oder Completion return address
IY+8,9  0x0000  oder Error return address
IY+10           Completion Code (wird vom Treiber gesetzt)

Zulässige Request Codes
0x00    Initialize Request     Keine Aktion
0x02    Assign Request         Keine Aktion
0x04    Open Request           Keine Aktion
0x06    Close Request          Keine Aktion
0x0A    Read Binary            Lesen von Datum/Zeit aus der RTC
0x0E    Write Binary           Setzen von Datum/Zeit in der RTC

Mögliche Completion Codes
0x80    Operation complete     Fehlerfreie Abarbeitung des Aufrufs
0x41    Invalid or inactive device
                               Keine RTC im System
0xC1    Invalid Request        Ungülter Request Code
0xC6    Data Transfer Error    Datenübertragung von oder zur
                               RTC fehlgeschlagen
```

Das im angegebenen Puffer zurück gelieferte (Datum/Zeit lesen) bzw. zu übergebende (Datum/Zeit setzen) 6-Byte-Datenfeld hat den Aufbau `YYMMDDhhmmss` im gepackten BCD-Format. D. h. zwölf Dezimalstellen werden in 6 Byte gespeichert und das Anwenderprogramm muss daraus das für die Anzeige oder Weiterverarbeitung notwendige Format selbst erzeugen.

Offset	Inhalt	Wert
0	YY Jahr	00-99
1	MM Monat	01-12
2	DD Tag	01-31
3	hh Stunde	00-23
4	mm Minute	00-59
5	ss Sekunden	00-59

Die Interpretation der zweistelligen Jahresangabe für ein bestimmtes Jahrhundert obliegt dabei dem Anwender.

N RomWBW Datenträger vorbereiten

Präparieren einer 128 MB CF-Karte mit dem Dienstprogramm **'FDISK80'**, welches unter einer in *RomWBW* enthaltenen CP/M-Version gestartet wurde. Die CF-Karte enthielt in ihrem früheren Einsatz ein Linux.

```
------------------------------------------------------------------------
B>fdisk80
FDISK80 for RomWBW, UNA, Mini-M68k, KISS-68030, SBC-188  ----
       Version 1.1-23 created 3-June-2023
                 (Running under RomWBW HBIOS)

HBIOS unit number [0..3]: 3
Capacity of disk 3:  (125M)  256000      Geom 03e81010
Nr  ---Type- A --     Start         End   LBA start  LBA count  Size
 1     Linux * 83    15:0:1   498:15:32        7680     247808  121M
 2     Linux   83     0:0:2     5:15:32           1       3071    1M
 3     Linux   83     6:0:1    14:15:32        3072       4608    2M
 4             00      *** empty ***
>>I
>>P
Nr  ---Type- A --     Start         End   LBA start  LBA count  Size
 1             00      *** empty ***
 2             00      *** empty ***
 3             00      *** empty ***
 4             00      *** empty ***
>>N1
Starting Cylinder (default 0): +1MB
Ending Cylinder (or Size= "+nnn"): +120MB
>>P
Nr  ---Type- A --     Start         End   LBA start  LBA count  Size
 1     FAT16   06     8:0:1   967:15:16        2048     245760  120M
 2             00      *** empty ***
 3             00      *** empty ***
 4             00      *** empty ***
>>T1
New type (in hex), "L" lists types: 2e
>>P
Nr  ---Type- A --     Start         End   LBA start  LBA count  Size
 1    RomWBW   2e     8:0:1   967:15:16        2048     245760  120M
 2             00      *** empty ***
 3             00      *** empty ***
 4             00      *** empty ***
>>W
Do you really want to write to disk? [N/y]: y
Okay
FDISK exit.

B>
------------------------------------------------------------------------
```

O RCBD auf Datenträger kopieren

Für diesen Vorgang ist ein Rechner (PC) erforderlich, der einen Kartenleser oder einen Anschluss für das gemäß Anlage N vorbereitete Speichermedium besitzt und das Linux Dienstprogramm **'dd'** ausführen kann. Das kann ein natives Linux System sein, ein Windows-PC mit installierter Cygwin-[15] oder MSYS2-Umgebung[16] oder ein sonstiges System, das Diskimages punktgenau auf einen externen Datenträger schreiben kann. Folgende Annahmen gelten für die Beispiele:

- Der externe Datenträger wird im Kopierrechner als `/dev/sdg` angesprochen
- Die *RomWBW*-Partition auf diesem Datenträger erscheint als `/dev/sdg1`

O.1 Berechnen der Blockadresse

Um das Image einer *UDOS*-Diskette oder eines *RCBD-ROM*-Lodaers in ein bestimmtes Slice in die *RomWBW*-Partition eines Datenträgers zu kopieren, ist die Berechnung der Blockadresse (Parameter `seek=x`), bei der das Slice beginnt, notwendig. Abhängig von der bei **'dd'** verwendeten Blocksize muss dazu die Slicenummer mit einem konstanten Faktor multipliziert werden.

- `bs=8M  Blockadresse = Slicenummer`
- `bs=1M  Blockadresse = Slicenummer * 8`
- `bs=1K  Blockadresse = Slicenummer * 8192`

O.2 Kopieren eines UDOS-Diskettenimages

Das Diskettenimage einer *UDOS*-Diskette ist immer 640 KiB groß und beginnt logisch bei Block 0 des Zielslices. Der folgende Befehl gilt unabhängig von der Diskettenart (Anwenderdiskette, Systemdiskette, mit oder ohne *RCBD-ROM*-Loader). Im folgenden Beispiel heiße die Imagedatei `RCBD-Sys-Z80.img` und soll in Slice 12 des Datenträgers platziert werden.

```
-------------------------------------------------------------------
$ dd if=RCBD-Sys-Z80.img of=/dev/sdg1 bs=8M count=1 seek=12
0+1 records in
0+1 records out
655360 bytes (655 kB, 640 KiB) copied, 0.262176 s, 2.5 MB/s
-------------------------------------------------------------------
```

[15] https://www.cygwin.com/
[16] https://www.msys2.org/

O.3 RCBD-ROM-Loader-only-Slice

Ein *RCBD-ROM*-Loader-Image ist zwischen 8 KiB und maximal 9,5 KiB groß und beginnt logisch bei Block 0 des Zielslices. Bei einem *RCBD-ROM*-Loader-only-Slice muss keine Rücksicht auf *UDOS*-Sektoren am Sliceanfang genommen werden, da es sich bei einem solchen Slice nicht um eine *UDOS*-Diskette handelt. Im folgenden Beispiel heiße die Imagedatei `rcbdloaderZ180.img` und soll in Slice 10 des Datenträgers platziert werden.

```
----------------------------------------------------------------------
$ dd if=rcbdloaderZ180.img of=/dev/sdg1 bs=8M count=1 seek=10
0+1 records in
0+1 records out
8424 bytes (8.4 kB, 8.2 KiB) copied, 0.0025865 s, 3.3 MB/s
----------------------------------------------------------------------
```

Ein *RCBD-ROM*-Loader-only-Slice kann nachträglich in eine *UDOS*-Diskette konvertiert werden, ohne dass der *RCBD-ROM*-Loader dabei verloren geht (*RCBD-ROM*-Loader-on-Disk). Dazu ist das betreffende Slice unter *RCBD-UDOS* lediglich als *UDOS*-Diskette zu formatieren, wobei die Option `rl=y` zu verwenden ist.

```
----------------------------------------------------------------------
rcbd:format [d=<drive>] [id='<max_24_char'] [s] rl=y
----------------------------------------------------------------------
```

O.4 RCBD-ROM-Loader-on-Disk

Soll ein *RCBD-ROM*-Loader-Image auf ein Slice mit einer existierenden *UDOS*-Diskette kopiert werden (*RCBD-ROM*-Loader-on-Disk), muss diese Diskette mit der Option `rl=y` des **'format'** Dienstprogramms formatiert worden sein und kann bereits Daten enthalten. Um keine Daten auf der Diskette zu überschreiben, darf beim Transfer das erste KiB der Imagedatei nicht mit in das Slice kopiert werden. Deswegen muss hier mit einer Blocksize von 1 KiB gearbeitet werden. Das Zielslice beginnt daher bei Blocknummer `Slice_Nummer*8192`. Dazu ist ein weiteres KiB für den übersprungenen Teil der Imagedatei zu addieren.

Im folgenden Beispiel soll die Imagedatei `rcbdloaderZ180.img` den vorhandenen *RCBD-ROM*-Loader auf der *UDOS*-Diskette in Slice 8 ersetzen/aktualisieren.

```
----------------------------------------------------------------------
$ dd if=rcbdloaderZ180.img of=/dev/sdg1 bs=1K count=10 \
                                          skip=1 seek=$((8*8192+1))
7+1 records in
7+1 records out
7400 bytes (7.4 kB, 7.2 KiB) copied, 0.0018995 s, 3.9 MB/s
----------------------------------------------------------------------
```

O.5 Update des RCBD-ROM-Loaders

Hat man ein lauffähiges *RCBD-UDOS*, kann man den *RCBD-ROM*-Loader auch aus dem laufenden System heraus aktualisieren. Dazu ist die neue Imagedatei mit XMODEM in eine Floppy- oder RAM-Disk des laufenden Systems zu kopieren. Der Dateitransfer sollte dabei im 128-Byte-Mode von XMODEM als Binärdatei durchgeführt werden. Bei TeraTerm ist dazu die Option `[ ]1k` im Dateiauswahldialog zu deselektieren.

Die hochgeladene Datei kann danach mit dem Dienstprogramm **'romwrite'** (11.6) in das Zielslice kopiert werden. Es ist unerheblich, ob das Zielslice ein *RCBD-ROM*-Loader-only-Slice oder eine mit der Option `rl=y` formatierte *RCBD-UDOS*-Diskette (*RCBD-ROM*-Loader-on-Disk) ist.

Wird im Zielslice ein früherer *RomWBW*-Loader erkannt, wird eine Meldung mit dem Label des gefundenen Loaders angezeigt, bevor der Kopiervorgang nach einer Sicherheitsabfrage gestartet wird.

```
---------------------------------------------------------------------
rcbd:xm -rb 1/newloader.img        ; Hochladen der neuen Imagedatei
XMODEM for RCBD 241216
File open - ready to receive
To cancel: Ctrl-X, pause, Ctrl-X
File size: 66 records (9k)
rcbd:md 2=4:63                     ; Zielslice bzw. -diskette Mappen
| Drv0 | Ram1 | Drv2 | Drv3 |
| 2.12 | 0:00 | 4:63 |      |
rcbd:romwrite d=2 newloader.img    ; Neuen RCBD-ROM-Loader schreiben
Found existing Boot Info Block labeled 'RCBD Z180 260120' on drive 2
Make sure reserved sectors exist on drive 2.
Write new RCBD-ROM-Loader? [yN] y
RCBD ROM Loader written
---------------------------------------------------------------------
```

Ende des Dokuments