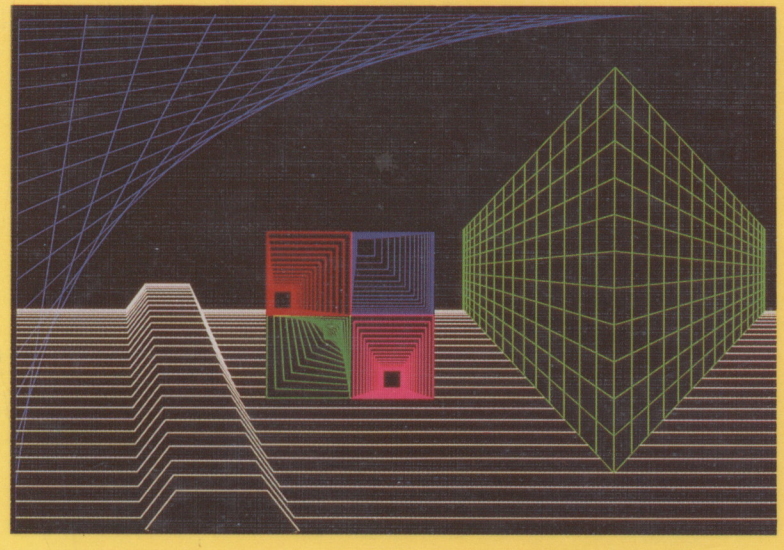
Franzis
Computer-Praxis

Klein
Die Prozessoren 68000 und 68008

Rechnerarchitektur und Sprache im NDR-KLEIN-Computer



Franzis'

Klein
Die Prozessoren 68000 und 68008

In der Reihe
Franzis Computer-Praxis
sind erschienen:

Busch, Basic für Aufsteiger
Busch, Basic für Einsteiger
Busch, Der sichere Einstieg in Pascal
Esders, Das Buch zum Apple II
Esders, Assembler-Programme zum Apple II
Feichtinger, Mit Computern steuern
Haugg, Software-Engineering und ihre Qualitätssicherung
Klein M./Klein R. D., Z-80-Applikationsbuch
Klein R.D., Basic-Interpreter
Klein R.D., Mit HEXMON Programme entwickeln
Klein R.D., Mikrocomputer Hard- und Softwarepraxis
Klein R.D., Mikrocomputer selbstgebaut und programmiert
Klein R.D., Mikrocomputersysteme
Klein R.D., Was ist Pascal
Link, Messen, Steuern und Regeln mit Basic
Merker, Hardware-Erweiterungen für den ZX 81
Piotrowski, IEC-Bus
Plate, Betriebssystem CP/M
Plate, Anwenderhandbuch CP/M-68K
Plate/Wittstock, Pascal: Einführung – Programmentwicklung – Strukturen
Pütz, Das große C64-Arbeitsbuch
Ruhland, DOS 3.3 – das Diskettenbetriebssystem des Apple II
Troitzsch, Mikrocomputer-Schaltungstechnik
Wunderlich, Erfolgreicher mit CBM arbeiten
Wunderlich, Erfolgreicher mit dem VC 64 arbeiten

Franzis Computer-Praxis

Rolf-Dieter Klein

Die Prozessoren 68000 und 68008

Rechnerarchitektur und Sprache im NDR-KLEIN-Computer

Mit 530 Abbildungen und 15 Tabellen

Franzis'

CIP-Kurztitelaufnahme der Deutschen Bibliothek

Klein, Rolf-Dieter:
Die Prozessoren 68000 und 68008: Rechnerarchitektur u. Sprache im NDR-KLEIN-Computer / Rolf-Dieter Klein. – München: Franzis; Luzern: Elektronik-Verlag, 1986.
(Franzis-Computer-Praxis)
ISBN 3-7723-7651-7

Satz: Franzis Technik GmbH, München
Druck: Wiener Verlag, A-2325 Himberg

Printed in Austria · Imprimé en Autriche

ISBN 3-7723-7651-7

Vorwort

In zunehmendem Maße dringen 16-Bit-Mikroprozessoren in die Technik ein und finden dort ihre Anwendung. Diese Mikroprozessoren der neuen Generation zeichnen sich durch ihre hohe Leistung und einen leicht erlernbaren Befehlssatz aus.

So finden die Prozessoren Anwendung in der Simulationstechnik, wie auch als Hilfsmittel zur graphischen Darstellung von Vorgängen der Natur. Aber auch im Einsatz bei der Erfassung von Meßwerten und deren Verarbeitung.

In diesem Buch wird der NDR-KLEIN-Computer als Hilfsmittel verwendet, um diese neue Technologie zu vermitteln. Dabei kann man mit diesem Computer sowohl 8-Bit-Technologie (Z80), als auch die 16-Bit-Technologie (68008/68000) wie auch die 32-Bit-Technologie (68020) kennenlernen. Der Computer ist modular aufgebaut und erlaubt es dadurch, unterschiedliche Prozessoren kennenzulernen. Durch das modulare Konzept kann er nicht veralten, denn durch neu hinzukommende Baugruppen wird er immer auf aktuellem Stand gehalten. Das Konzept eignet sich sehr gut für die Ausbildung, bei der es darauf ankommt, aktuelle Informationen zu vermitteln.

Der moderne Graphik-Prozessor im NDR-KLEIN-Computer ermöglicht eine sehr leistungsfähige und sogar bewegte Graphik.

Alle zum Aufbau des Rechners notwendigen Schaltpläne und Programmlisten stehen dem Leser dieses Buches zur Verfügung. Somit ist der Selbstbau des Computers möglich. Einen großen Raum nimmt die Programmierung des Rechners ein, so lernt man neben der Assemblerprogrammierung auch den Umgang mit höheren Programmiersprachen wie Pascal und Gosi (Logo-ähnlich). Aber auch der Anschluß eines Massenspeichers und die Nutzung eines professionellen Betriebssystems (CP/M-68k) wird ausführlich behandelt.

Sowohl der Anfänger als auch der Profi finden wichtige Informationen zu den Prozessoren 68000 und 68008 in diesem Werk.

Das Buch setzt allgemeine Grundkenntnisse der Mikrocomputertechnik voraus.

Rolf-Dieter Klein, München

Wichtiger Hinweis

Die in diesem Buch wiedergegebenen Schaltungen und Verfahren werden ohne Rücksicht auf die Patentlage mitgeteilt. Sie sind ausschließlich für Amateur- und Lehrzwecke bestimmt und dürfen nicht gewerblich genutzt werden*).
Alle Schaltungen und technischen Angaben in diesem Buch wurden vom Autor mit größter Sorgfalt erarbeitet bzw. zusammengestellt und unter Einschaltung wirksamer Kontrollmaßnahmen reproduziert. Trotzdem sind Fehler nicht ganz auszuschließen. Der Verlag und der Autor sehen sich deshalb gezwungen, darauf hinzuweisen, daß sie weder eine Garantie noch die juristische Verantwortung oder irgendeine Haftung für Folgen, die auf fehlerhafte Angaben zurückgehen, übernehmen können. Für die Mitteilung eventueller Fehler sind Autor und Verlag jederzeit dankbar.

---

*) Bei gewerblicher Nutzung ist vorher die Genehmigung des möglichen Lizenzinhabers einzuholen.

Inhalt

1 Der NDR-KLEIN-Computer

Der 68000/8 Computer wird aus den Baugruppen des NDR-KLEIN-Computers zusammengestellt, aus welchen sich viele Computerkonfigurationen ergeben. Im Kapitel 7 werden alle für dieses Buch interessanten Baugruppen ausführlich beschrieben, so daß der Selbstbau möglich ist. In den folgenden Abschnitten wird jedoch davon ausgegangen, daß die Baugruppen bereits fertig aufgebaut sind. Wer also die Baugruppen erst noch zusammenbauen muß, der sollte im Kapitel 7 anfangen zu lesen.

Den Computer kann man unterschiedlich zusammensetzen, jedoch entspricht der Gesamtaufbau *Abb. 1.1.*

Über eine Tastatur werden Befehle an den Computer gegeben. Auf dem Bildschirm erfolgt die Ausgabe. Daten und Programme kann man auf einem Cassettenrecorder speichern und davon laden. Alternativ kann man auch ein oder mehrere Floppy-Laufwerke anschließen. Ein Floppy-Laufwerk ist zuverlässiger und schneller, aber auch teurer als ein Cassettenrecorder. Daher sollte man ruhig zunächst einmal mit dem Recorder anfangen und erst später ein Laufwerk dazukaufen.

1.1 Übersicht über die Baugruppen des NDR-KLEIN-Computers

Für den NDR-KLEIN-Computer gibt es weit mehr Baugruppen, als wir momentan für unseren Bedarf brauchen. Zunächst einmal gibt es eine Reihe von CPU-Baugruppen:

- SBCII
- Z80 Vollausbau-CPU
- CPU68K
- CPU68K/16

Zum einen gibt es CPU-Karten mit dem Z80. Hier ist zunächst die SBCII-Baugruppe zu nennen. Sie besitzt einen Z80 und außerdem bereits 4K RAM sowie 8K EPROM. Damit lassen sich kleinere Steuerungsaufgaben lösen, und man kann damit in die Programmierung des Z80 einsteigen. Der Speicher dieser Baugruppe läßt sich aber nicht ohne weiteres voll ausbauen. Wenn man das will, so verwendet man die Vollausbau-CPU mit dem Z80, die direkt 64K Byte Speicher ansprechen kann. Beide Baugruppen sind in dem Buch „Mikrocomputer selbstgebaut und programmiert" ausführlich beschrieben.

Wir wollen uns hier aber nicht mehr mit dem Z80 aufhalten, sondern den 68000/8 verwenden. Dafür gibt es zwei unterschiedliche Karten. Die Baugruppe CPU68K enthält den Mikroprozessor 68008, der einen gemultiplexten 8-Bit-Datenbus besitzt. Sonst ist er aber ein vollwertiger 16-Bit-Mikroprozessor, sein Befehlssatz ist mit dem der 68000 CPU identisch. Die CPU68K/16 enthält dann den „echten" 68000 mit 16-Bit-Datenbus, so daß man auch diese CPU verwenden kann. Man sollte das aber nur dann tun, wenn

man die höhere Geschwindigkeit der 68000 CPU gegenüber der 68008 CPU auch wirklich ausnützen will, denn durch den 16 Bit breiten Datenbus benötigt man jede Speicherbaugruppe doppelt, und das System wird dadurch teurer. Man gewinnt etwa den Faktor zwei an Geschwindigkeit, bei gleicher CPU-Frequenz. Der 68000 ist aber mit höheren Taktraten lieferbar, so daß sich daraus nochmals eine Möglichkeit zur Geschwindigkeitssteigerung ergibt. Eine weitere Alternative stellt die 68010 CPU dar, die nochmals geringfügige Verbesserungen bietet. Den NDR-KLEIN-Computer kann man natürlich auch nachträglich aufrüsten, und man muß sich nicht gleich für eine endgültige Konfiguration entscheiden.

In den späteren Kapiteln wird vorzugsweise der 68008 Prozessor verwendet. Da sich in der Bedienung fast keine Unterschiede ergeben, ist das aber nicht weiter von Bedeutung.

Dann gibt es natürlich auch eine Reihe von unterschiedlichen Speicherbaugruppen:
- ROA64
- DYN64/256
- BANK/BOOT
- BANKSEL

Die Baugruppe ROA64 ist die wichtigste Speicherbaugruppe. Sie verwendet statische 8Kx8 RAMs und kann auch gemischt mit 8Kx8 EPROMs bestückt werden. Diese Karte bildet die Basis unseres Computers. Sie kann maximal 64K Byte Speicher aufnehmen.

Wer den Einsatz dynamischer RAMs nicht scheut, kann mit der Baugruppe DYN64/256 solche Speicher einsetzen. Sie sind im Durchschnitt preisgünstiger als statische Speicher und auch mit einer höheren Kapazität lieferbar. Die Speicher haben auch einen Nachteil. So kann die CPU nicht mit ihrer maximalen Geschwindigkeit auf die RAMs zugreifen, denn die RAMs müssen ständig aufgefrischt (refresh) werden, damit sie die Information nicht verlieren. Und während des Auffrischens kann die CPU nicht zugreifen. Auf der DYN64/256 Baugruppe kann man wahlweise 64K dynamische Speicher oder 256K dynamische Speicher einsetzen und erhält damit 64K Byte oder 256K Byte pro RAM-Karte. Unsere CPU-Karte mit dem 68008 kann maximal 1 Mega-Byte adressieren, die 68000-CPU-Karte kann 2Mega-Byte adressieren.

Die Baugruppe BANK/BOOT schließlich wird verwendet, um Speicherbereiche aus dem normalen Adreßbereich auszublenden. Dies verlangt z. B. das Disketten-Betriebssystem CP/M-68k ab Adresse 0.

Die Baugruppe BANKSEL dient dem gleichen Zweck und kann ebenfalls verwendet werden, sie ist jedoch nicht für die 8Kx8 EPROMs vorbereitet. Wie das genau funktioniert, ist im Kapitel 6, Betriebsysteme und Floppy, beschrieben.

Damit sind wir bei den Peripherie-Baugruppen angelangt:
- KEY – AD16x8
- GDP64 – AD1x10
- CAS – DA2x8
- FLO2 – UHR
- IOE – SER
- PROMMER

Die Baugruppe KEY dient dem Anschluß einer Tastatur. Die Baugruppe GDP64 beinhaltet einen Graphic-Display-Prozessor, der mit einem eigenen 64K Byte Speicher

arbeitet und dem Anschluß eines Bildschirms dient. Diese Baugruppe bietet sehr schnelle Graphikmöglichkeiten und eine Auflösung von 512 x 256 Bildpunkten. Dabei können vier solcher Bildseiten abgespeichert werden.

Die Baugruppe CAS stellt die Verbindung zum Cassettenrecorder dar und enthält die dafür notwenige Logik.

Auf der Baugruppe FLO2 befindet sich ein sogenannter Floppy-Controller, der den Anschluß von unterschiedlichen Floppy-Laufwerken ermöglicht.

Mit der Baugruppe IOE besitzt man ein universelles Experimentierfeld, auf dem sich eigene Schaltungen aufbauen lassen. Außerdem stellt diese Karte eine Druckerschnittstelle nach Centronics zur Verfügung.

Mit der Baugruppe PROMMER kann man EPROMs selbst programmieren.

Die Baugruppen AD16x8 und AD1x10 sind Analog/Digital-Umsetzer, mit denen man z.B. Spannungen messen kann. Sie wandeln also analoge Spannungen in digitale Größen um.

Mit der Baugruppe DA2x8 kann man einen digitalen Wert in einen analogen umsetzen.

Wie der Name der UHR-Baugruppe bereits verrät, wird unser System dadurch um eine batteriegepufferte Uhr erweitert. Diese Uhr liefert neben der Uhrzeit auch Wochentag, Monat und Jahr.

Die Baugruppe SER dient der Erzeugung eines seriellen Signals, mit dem man ebenfalls einen Drucker bedienen kann, sie versetzt uns aber auch in die Lage, ein Modem zur Kopplung unseres Rechners über Telefon. anschließen zu können.

1.2 Der 68008-Computer

Was braucht man für das Minimalsystem mit dem 68008?

Zunächst einmal eine Spannungsversorgung. Man kann dazu die Baugruppe POW5V verwenden, diese liefert einen Strom, der für den kleinen Ausbau reicht. Oder man verwendet ein großes Netzteil, diese Spannungsversorgung sollte eine Spannung von 5V bei einigen Ampere Strom (3A bis 6A) liefern. Um auch für spätere Erweiterungen z.B. mit der Floppy gerüstetet zu sein, sollte sie ferner eine +12V Ausgangsspannung besitzen, mit ca. 1A bis 2A Strom. –12V benötigt man für die serielle Schnittstelle und –5V für den A/D-Umsetzer AD1x10. Wer diese Baugruppen einsetzen will, plant dies am besten gleich mit. Bei diesen Spannungen genügt 1A bei weitem. Für den EPROM-Programmierer wird noch eine eigene Spannung benötigt, die bei 26V bzw. 22V liegen sollte. Da man sich aber auch mit einem Spannungswandler behelfen kann, (siehe Kapitel PROMMER) muß dies kein wesentlicher Schwerpunkt bei der Netzteilentscheidung sein.

Ferner benötigt man natürlich einen Bildschirm, das kann ein Fernseher sein, zu dem man einen HF-Modulator benötigt, oder besser ein Monitor, mit dem sich ein schärferes Bild ergibt. Außerdem braucht man eine Tastatur, die den ASCII liefert (siehe Kapitel zur KEY-Baugruppe).

Als Basis für den Aufbau des Computers benötigt man die BUS-Baugruppe, auf die alle Baugruppen gesteckt werden. Es gibt sie in unterschiedlichen Versionen. Für

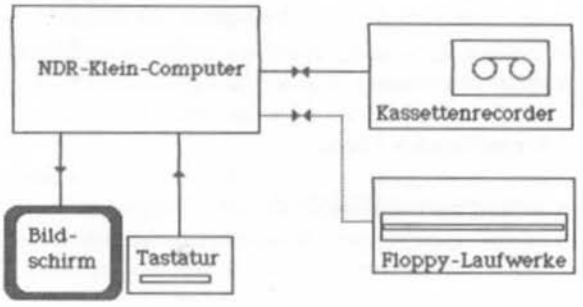


Abb. 1.1 Der NDR-KLEIN-Computer im Gesamtaufbau

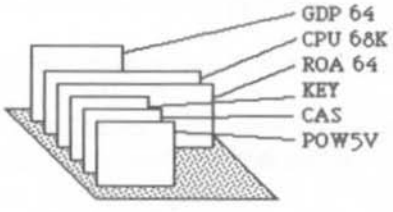
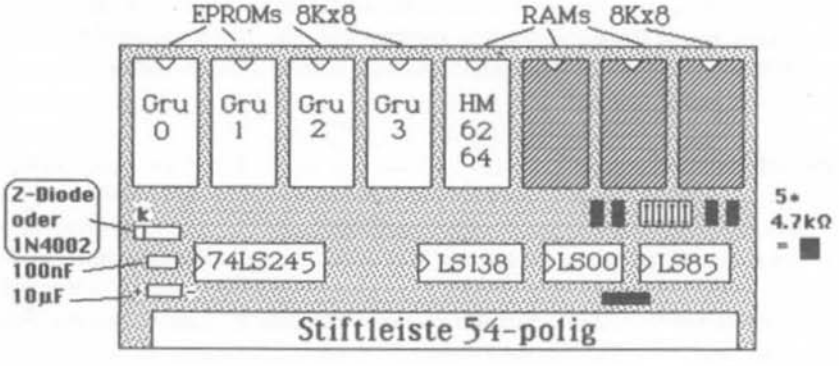


Abb. 1.2.1
Minimalsystem
mit dem 68008



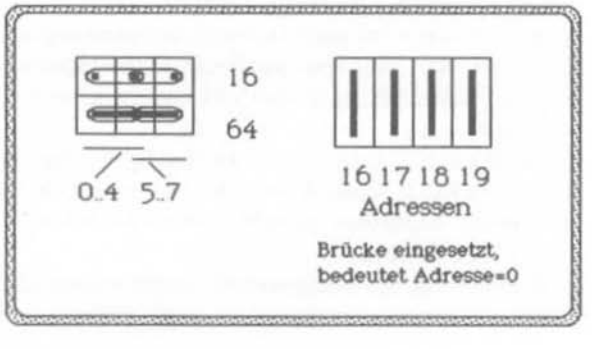


Abb. 1.2.2
Einstellung der Brücken
auf der ROA64

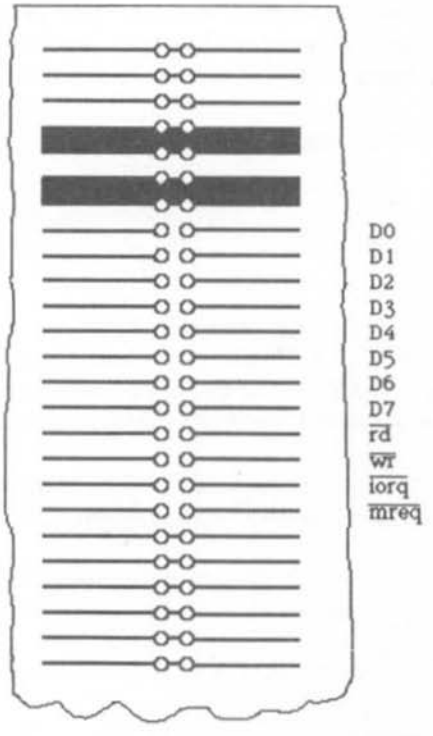


Abb. 1.3.1 Lötseite der Busbaugruppe mit Änderungen für den 68000

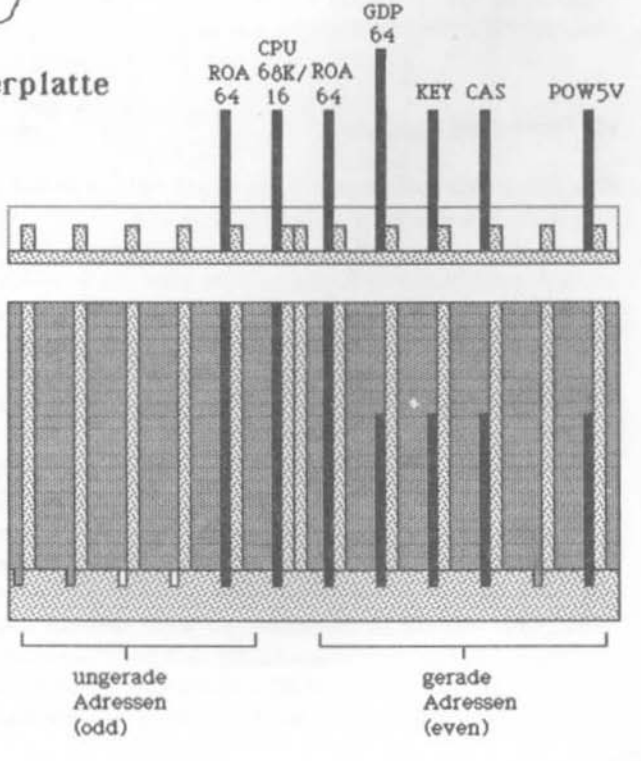


Abb. 1.3.2 Anordnung der Baugruppen auf dem Bus, beim Betrieb mit dem 68000

unsere Versuche reicht die kürzere Version, von der dieses Buch ausgeht. Wenn man aber viel ausbauen will, so kann man sich gleich einen längeren Bus besorgen, aber auch den kurzen Bus kann man nachträglich noch ausbauen, indem man einen weiteren Bus einfach dazusetzt.

Wir benötigen folgende Baugruppen-Zusammenstellung:

1. CPU 68K mit dem 68008-Prozessor
2. Die Baugruppe ROA64
3. Die Baugruppe KEY
4. Die Baugruppe GDP64
5. Die Baugruppe CAS
6. Die Baugruppe POW5V, falls kein externes Netzteil verwendet wird.
7. Einen EPROM-Satz mit dem Grundprogramm 68K (vier EPROMs mit 8Kx8 Bytes).
8. Mindestens einen RAM-Baustein HM6264 mit 8Kx8 Bytes. Besser ist es, gleich zwei RAMs vorzusehen, denn mit einem Baustein lassen sich nicht alle Versuche dieses Buches durchführen.

Abb. 1.2.1 zeigt die Anordnung des Minimalsystems. Die Reihenfolge der Baugruppen auf dem Bus spielt keine Rolle, jedoch müssen natürlich die Baugruppen richtig herum eingesteckt werden.

Die Baugruppe ROA64 wird mit den EPROMs und dem RAM gemäß der Abbildung bestückt. *Abb. 1.2.2* zeigt die notwenigen Brücken auf der ROA64, um sie für den Adressbereich des Grundprogramms einzustellen.

Damit ist der Computer betriebsbereit. Nach dem Einschalten muß sich das Grundprogramm auf dem Bildschirm melden.

1.3 Der 68000-Computer

Wer gleich den großen 68000 einsetzen will, muß die große BUS-Baugruppe verwenden, denn der Bus wird in der Mitte geteilt.

Der 68000 unterscheidet sich vom 68008 nicht nur durch den breiteren Datenbus, sondern auch in seiner Bauform. Also Vorsicht beim Einsetzen der CPU in die Fassung.

Die große BUS-Baugruppe ist in der Mitte mit einer doppelten Lochreihe versehen. Diese Lochreihen sind normalerweise durchverbunden. Man muß zur Vorbereitung für den 68000 den Bus in der Mitte aufteilen, dazu werden 12 Leitungen getrennt. *Abb. 1.3.1* zeigt ein Schema dazu, die Busbaugruppe wird von der Lötseite aus betrachtet. Die Versorgungsleitungen bleiben durchverbunden, darauf folgen 12 Leitungen , die unterbrochen werden, die restlichen können verbunden bleiben. Auf der Bestückungsseite wird anstelle der sonst üblichen einreihigen Buchsenreihe eine doppelreihige Buchsenreihe eingelötet, denn die CPU-Baugruppe besitzt eine entsprechende doppelreihige Stiftleiste. Durch die Trennung ist es möglich, für die beiden Seiten eine unterschiedliche Signalbelegung für den Datenbus D0 bis D7 sowie für die Steuerleitungen -RD, -WR, -IORQ und -MREQ zu erreichen. Es wird also ein doppelt (16 Bit) breiter Datenbus verfügbar. Die Datenleitungen D0 bis D15 des Prozessors werden auf die beiden Hälften verteilt. Eine Aufteilung der Steuerleitungen ist nötig, da die CPU in der Lage ist, auch wahlweise nur auf eine der beiden Hälften zuzugreifen.

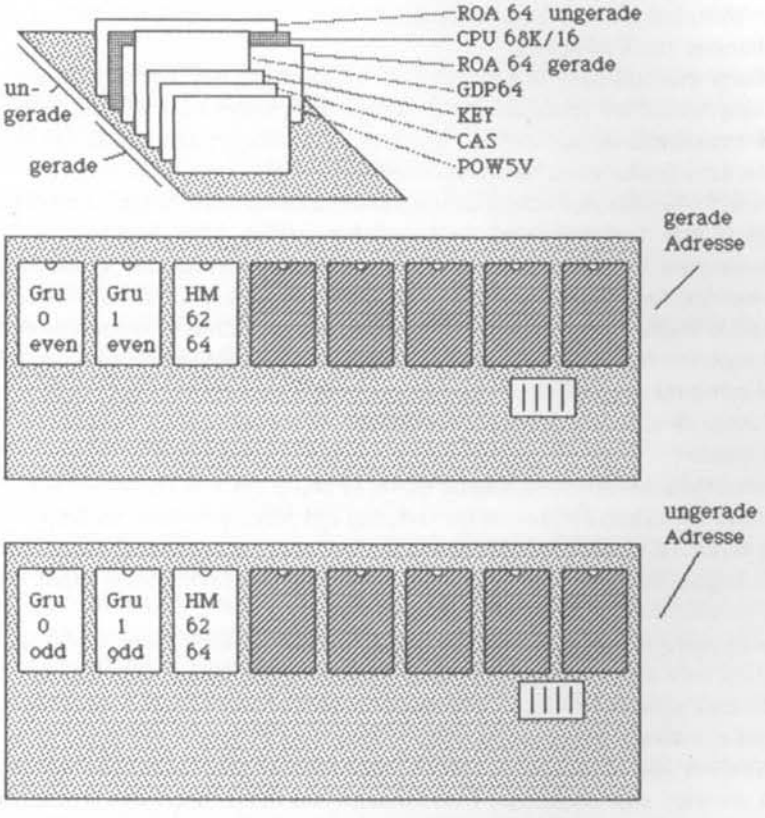


Abb. 1.3.3 Die Anordnung der EPROMs mit dem Grundprogramm beim 68000-System

Durch die unterschiedliche Funktion der beiden Hälften ist die Lage der Baugruppen nicht mehr gleichgültig. *Abb. 1.3.2* zeigt ein Schema. Die CPU-Baugruppe sitzt in der Mitte des Busses. Links davon befinden sich die ungeraden Adressen und rechts die geraden. Man benötigt jetzt zwei ROA64 Baugruppen, um beide Bushälften mit Daten zu versorgen. Die Peripheriebaugruppen, die durch das Grundprogramm unterstützt werden, befinden sich auf der „geraden" Seite. Die „ungerade" Seite steht eigenen Erweiterungen zur Verfügung.

Man kann nun mit dem Bus doppelt soviel Speicher und Peripherie adressieren, da die Adresse A0 nur zur Auswahl der Bushälfte verwendet wird und die Adressen A1 bis A20 des Prozessors an die mit A0 bis A19 beschrifteten Leitungen des Busses angeschlossen sind (siehe auch Kapitel CPU68K/16).

Abb. 1.3.3 zeigt die Anordung der EPROMs auf den ROA-Baugruppen. Die EPROMs sind mit 0 und 1 sowie even und odd beschriftet. Man benötigt jetzt außerdem mindestens zwei RAM-Bausteine. Jede Speichererweiterung muß symmetrisch durchgeführt werden, man braucht also pro Erweiterung immer zwei RAM-Bausteine.

Die Brücken sind alle eingesetzt, am Adreßbereich ändert sich ansonsten nichts. Das Grundprogramm für den 68000 ist nicht identisch mit dem 68008, alle Peripherieadressen sind mit zwei multipliziert. Das gilt auch für Versuche aus dem Buch; man muß bei der CPU68K/16 alle angegebenen Peripherie-Adressen in den Programmen mit zwei multiplizieren.

Hier nochmals der Hinweis: Die CPU68K/16 sollte nur von Fortgeschrittenen verwendet werden, die schon Erfahrung im Umgang mit Mikroprozessoren haben und die die Vorteile dieser CPU auch wirklich ausnützen wollen; alle anderen sollten sich mit der CPU68K begnügen. Wie bereits gesagt, lassen sich mit der 68008 exakt die gleichen Versuche ausführen, wie mit der großen Schwester.

Übrigens wird die 68000 Familie mit diesem Prozessor nicht abgeschlossen sein, denn schon naht eine weitere Neuerung: die 68020 CPU. Sie besitzt einen 32 Bit breiten Datenbus und außerdem einen kleinen internen Speicher, CACHE genannt, in dem sich die CPU die letzten Befehle merkt und die Befehle von dort direkt holt, wenn der Befehl dort enthalten ist. Damit ist natürlich eine bedeutende Geschwindigkeitssteigerung möglich. Es wird aber noch eine Weile dauern, bis die CPU für den Normalverbraucher verfügbar sein wird. Mit dem NDR-KLEIN-Computer kann man aber auch diese CPU verwenden, das ist sicher für die Industrie ebenso wie für die Universitäten interessant. Das Grundprogramm ist für die CPU schon vorbereitet.

2 Das Grundprogramm 68K

Um einen Computer betreiben zu können, ist außer der Hardware auch die entsprechende Software erforderlich. Gleich nach dem Einschalten muß diese Software – in welcher Form auch immer – aktiv werden. Die einfachste Form eines solchen Programms stellt ein sogenanntes Monitorprogramm dar.

Monitorprogramme sind mehr oder weniger komfortabel. Die minimalen Funktionen, die es aber können muß, sind:

1. Eingabe des Programms
2. Starten des Programms

Diese Funktionen genügen natürlich nicht, um Programme bequem entwickeln zu können; dazu benötigt man noch weitere Funktionen:

3. Speicherinhalte ansehen
4. Einzelschritte ausführen

Bei den meisten einfachen Monitorprogrammen werden die Programme in sedezimaler Form eingegeben, das ist eine Art der Eingabe von langen Zahlenketten, die sehr unanschaulich und entsprechend fehlerträchtig ist.

Wir erkennen schon, um sinnvoll Programme entwickeln zu können, ist so ein Monitorprogramm wesentlich zu erweitern. Da unser „Monitorprogramm" sehr komfortabel ist und grundsätzliche Neuerungen bringt, wurde es „Grundprogramm" genannt.

Durch unser Grundprogramm verfügen wir über einen komfortablen Texteditor, mit dem man die Programme eingeben kann. Dabei erfolgt die Eingabe im sogenannten Assemblercode, das ist eine symbolische Sprache, die man sich leichter merken kann, als den Maschinencode. Ein Übersetzungsprogramm, der Assembler, der ebenfalls im Grundprogramm enthalten ist, übernimmt die Aufgabe der Übersetzung in den Maschinencode.

Der Texteditor ermöglicht es aber auch, Programme in beliebigen Programmiersprachen und sogar Briefe einzutippen. Das Grundprogramm kann durch die höheren Programmiersprachen Pascal oder Gosi (ähnlich Logo) erweitert werden. Dann sind wir in der Lage, auch in diesen Sprachen zu arbeiten. Was es damit auf sich hat, werden wir noch nach und nach kennenlernen.

2.1 Das Menue

Jetzt wird es aber höchste Zeit, den Computer einzuschalten und das Grundprogramm kennenzulernen.

Nach dem Einschalten, falls alle Vorbereitungen gemäß Kapitel 1 getroffen wurden, erscheint auf dem Bildschirm zunächst die Copyright-Meldung. Kurze Zeit später ist dann das Grundmenue zu sehen (*Abb. 2.1.1*). Im unteren linken Feld blinkt der

sogenannte Cursor, der angibt, daß der Computer eine Eingabe von der Tastatur erwartet. An die blinkende Stelle wird dann bei einer Eingabe das nächste Zeichen geschrieben.

Drücken wir jetzt einmal irgendeine Taste, z.B. die Taste „A". An der Cursor-Position erscheint dann das „A", der Cursor rückt um ein Zeichen nach rechts. Gibt man nun einen weiteren Buchstaben ein, so verschwindet die Eingabe und der Cursor blinkt wieder in dem sonst leeren Feld, wie anfangs auf der linken Seite. Die Eingabe wurde vom Grundprogramm nicht verstanden.

Was sehen wir im ersten Menue? Das Menue enthält fünf Menuepunkte. An erster Stelle steht „1=aendern", damit ist gemeint, eine Eingabe einer „1" ruft die Funktion „aendern" auf. Mit 2 die Funktion „starten" und mit 3 die Funktion „ansehen", 4 ist für die Funktion „Symbole" und W für die Funktion „weiter" vorgesehen.

Wir wollen erst mal einen groben Rundgang unternehmen, bevor wir uns den einzelnen Menuepunkten widmen. Dazu wollen wir im Menue „weiter" und drücken die Taste „W". Normalerweise erscheint beim Druck auf eine Buchstabentaste ein kleiner Buchstabe. Soll ein großer Buchstabe eingegeben werden, so ist – wie bei einer Schreibmaschine – die (Umschalt-)Taste „SHIFT" zu betätigen und dann gleichzeitig der Buchstabe einzugeben. Das Grundprogramm versteht aber Groß- und Kleinbuchstaben gleichermaßen, daher spielt die Schreibweise jetzt keine Rolle. Wenn man die Taste „W" gedrückt hat und der Buchstabe „w" auf dem Bildschirm im Eingabefeld erschienen ist, tut sich zunächst nichts weiter. Das Grundprogramm erwartet eine Bestätigung der Eingabe. Diese Bestätigung geschieht mit der Taste CR. CR steht für Carriage Return und bedeutet soviel wie Wagenrücklauf. Der Begriff ist der Schreibmaschinentechnik entlehnt. Auf der Computertastatur ist die Taste manchmal mit CR beschriftet, manchmal aber auch nur mit einem gewinkelten Pfeil oder mit der Bezeichnung „RETURN".

Wenn man jetzt diese Taste drückt, so erscheint das nächste Menue auf dem Bildschirm. Hier finden wir den schon früher erwähnten Texteditor, der mit „1" angesprochen werden kann, dann das Übersetzungsprogramm „2=Assembler". Ferner gibt es dort noch ein Menue mit dem Namen „3=Bibliothek", das hinführt zur Verwendung anderer, nicht im Grundprogramm vorhandener Sprachen und Programme. Mit „4=Optionen" gelangt man in ein Spezialmenue, von dem aus man sehr unterschiedliche Voreinstellungen durchführen kann. Wir gehen aber im Hauptmenue weiter und geben erneut die Folge „W" und „CR" ein. Unser Speichermenue wird sichtbar. CAS steht für die Baugruppe CAS, die einen Cassettenrecorder als Speicher einsetzt. Damit ist es möglich, Programme und Daten zu laden, zu speichern oder zu prüfen. Mit dem Menuepunkt „4=Floppy Start" kann man ein Floppy-Laufwerk als Speicher bedienen; dies wird in einem der späteren Kapitel beschrieben.

Wir gehen wieder „weiter" und gelangen in ein weiteres Menue. Die ersten beiden Menuepunkte dienen der Arbeit mit der Baugruppe PROMMER. Mit dieser Einrichtung ist es möglich, selbst EPROMs mit Programmen oder Daten zu laden, bzw. den Inhalt von EPROMs in den Speicher zu laden. Der Punkt „4=Text drucken" dient dazu, Texte, die mit dem Editor eingegeben wurden, direkt auf einen Drucker auszugeben. Der Computer läßt sich so als Textverarbeitungssystem, z. B. zum Schreiben von Briefen, einsetzen. Mit „3=Speicherbereiche" kann man sich über die aktuelle Lage von RAM-Plätzen informieren. Wir probieren diese Möglichkeit gleich einmal aus, wir geben also

Abb. 2.1.1 So sieht das Grundmenue aus

Abb. 2.1.2 Speicherbereiche beim Ausbau mit 16 KByte

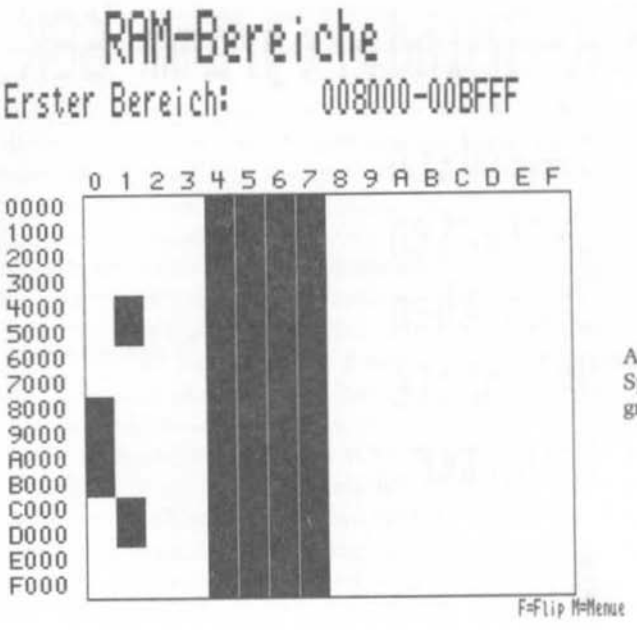


Abb. 2.1.3
Speicherbereiche, bei größerem Ausbau

Abb. 2.1.4
Die verschiedenen Menues beim Grundprogramm

„3" und zur Bestätigung „CR" ein. *Abb. 2.1.2* erscheint auf dem Bildschirm. Mit „Erster Bereich:" wird der erste zusammenhängende RAM-Bereich angegeben. Hier reicht er von der Adresse 008000 bis 00BFFF. Die Angabe des Zahlenwertes geschieht im sedezimalen Zahlensystem. In dem unteren Feld werden alle vorhandenen RAM-Bereiche als Flächen ausgegeben. Wie unschwer zu erkennen ist, gibt es in unserem Fall nur diesen einen Speicherbereich. Das Bild kann auch anders aussehen. *Abb. 2.1.3* zeigt ein weiteres Beispiel.

Zur Erkennung der RAM-Speicherbereiche wird ein kleiner Speichertest durchgeführt. Dieser Test ist allerdings kein vollständiger und sicherer Speichertest, dennoch eine gute Hilfe, um Fehler z.B. bei der Adressierung zu finden. Der Bereich F0000 bis FFFFF ist ganz reserviert und wird nie auftauchen; dort liegen die Adressen der Peripheriebaugruppen.

Der angegebene „erste Bereich" ist wichtig, weil das Grundprogramm den sogenannten Stack an das Ende dieses Bereichs legt. An das Ende des ersten Bereiches darf man deshalb nie Programme legen. Wie man die genaue Grenze bestimmt, werden wir später noch sehen.

Wie wir wieder ins Menue gelangen, entnehmen wird dem Hinweis: „F=Flip M=Menue". Wenn man die Taste „M" drückt, so gelangt man sofort wieder in das erste Grundmenue. Die Bestätigung per „CR" ist diesmal nicht erforderlich. CR muß man immer dann drücken, wenn zusätzlich ein blinkender Cursor sichtbar ist, das ist hier nicht der Fall. Wenn man anstelle der Taste „M" die Taste „F" gedrückt hat, so flimmert der Bildschirm plötzlich; wozu das gut ist, lernen wir im Kapitel 3.

Nun aber wieder weiter in den Menues. Dazu muß man jetzt zuerst einmal das Menue mit dem Speicherbereich anwählen, da man beim Verlassen des Menuepunktes „Speicherbereiche" wieder in das erste Menue zurückgelangt ist.

Mit „W" und „CR" kommen wir in das nächste Menue. Die ersten beiden Menuepunkte sind primär für den Test von Baugruppen gedacht. Man kann damit zum Beispiel feststellen, ob ein Baustein auch wirklich die richtigen Informationen liefert oder nicht. Der Punkt „3=Einzelschritt" ist einer der wichtigsten Menuepunkte. Man kann damit Programme Schritt für Schritt ausführen und Registerinhalt etc. kontrollieren.

Nach diesem kleinen Orientierungsausflug wollen wir uns jetzt ausführlicher mit den einzelnen Menuepunkten beschäftigen. *Abb. 2.1.4* zeigt eine Übersicht aller Menues.

2.1.1 Das Grundmenue

Fangen wir beim ersten Menue an. Der Menuepunkt „1=aendern" dient dem Ändern von Speicherinhalten. Man kann sich aber auch kleine Speicherbereiche ansehen.

1=aendern

Sehen wir und das Ganze einfach mal an. Also vom ersten Menue aus gehts mit „1" und dann „CR" los. Jetzt wird *Abb. 2.1.5* auf dem Bildschirm erscheinen. Der Cursor blinkt neben der Stelle „Adr:". Das Grundprogramm erwartet nun die Eingabe einer Speicheradresse. Diese Eingabe kann auf verschiedene Art erfolgen; das Grundprogramm versteht mehrere Zahlensysteme, und es besitzt die Fähigkeit, arithmetische Rechnungen direkt auszuführen. Ferner kann man auch Namen eingeben, die bereits definiert sind und den Wert einer Adresse verkörpern. Doch halt, nicht alles auf einmal.

Beginnen wir mit den Zahlensystemen.

Zunächst einmal sind die Zahlen, die einfach so ohne besondere Maßnahmen eingetippt werden, dezimal. Als Beispiel tippen wir folgende Ziffernfolge ein:
1 0 2 4

Die Zahl erscheint auf dem Bildschirm. Wenn man sich vertippt hat, so kann man den Wert auch noch korrigieren. Durch Drücken der Taste DEL, wird das zuletzt eingebene Zeichen gelöscht. Drückt man dann nochmals die Taste DEL, so verschwindet auch das vorletzte Zeichen usw. Man kann auch die Tasten CTRL und dann dazu die Taste H drücken oder die Backspace-Taste (Pfeil links, oder BS). Es wurden mehrere Möglichkeiten zur Verfügung gestellt, da die Belegung bei den Tastaturen unterschiedlich ist. So findet man mindestens einen bequemen Weg zum Löschen.

Steht nun endgültig der richtige Wert auf dem Bildschirm, so bestätigen wir unsere Eingabe in der gewohnten Weise per „CR". Die *Abb. 2.1.6* erscheint. Jetzt stehen die Werte 000400:5A A5 80 01 auf dem Schirm. Der erste Zahlenwert steht für die angewählte Adresse, die hier im sedezimalen Zahlensystem ausgegeben wird. Die dezimale Zahl 1024 entspricht nämlich der sedezimalen Zahl 400. Danach wird der Inhalt von vier aufeinanderfolgenden Speicherzellen ausgegeben. 5A ist der Wert im sedezimalen Zahlensystem, den die Speicherzelle an der angegebenen Adresse besitzt. A5 steht in der nächsten, 80 in der dritten Zelle und 01 in der vierten Speicherzelle.

Der Cursor blinkt in dem darunterliegenden Feld. Der Computer wartet nun auf eine neue Eingabe. Man könnte jetzt den Inhalt der Speicherzelle verändern, doch das versuchen wir besser nicht, denn bei der angewählten Speicherzelle handelt es sich um ein Stück Grundprogramm, und das sitzt im EPROM, es läßt sich also nicht verändern. Um die Änderungsmöglichkeit doch noch ausprobieren zu können, geben wir eine neue Speicheradresse an. Dazu tippt man den Buchstaben „R" ein und drückt dann die Taste „CR". Wieder können wir eine Adresse eingeben. Diesmal wollen wir die sedezimale Adresse 9800 nehmen, sie liegt im RAM-Speicher. In der Grundausbaustufe des Rechners verfügen wir über einen RAM-Speicher von sedezimal 8000 bis 9FFF, wenn wir einen zweiten RAM-Baustein eingesetzt haben von 8000 bis BFFF. Diesen Bereich müssen wir mit dem Grundprogramm teilen. Es legt von Adresse 8000 bis ca. 8A4E einen ständigen Arbeitsspeicher an. Damit noch nicht genug, ab ca. 8A4E beginnt noch ein variabel langer Bereich für die bereits erwähnten Namen, die man sich definieren kann. Man sollte deshalb zunächst mal den ganzen Bereich 8000 bis 8FFF meiden.

Wenn man sedezimale Zahlen eingeben will, so muß man dies dem Grundprogramm durch das Voranstellen eines Dollarzeichens mitteilen. Wir geben also ein: $ 9 8 0 0

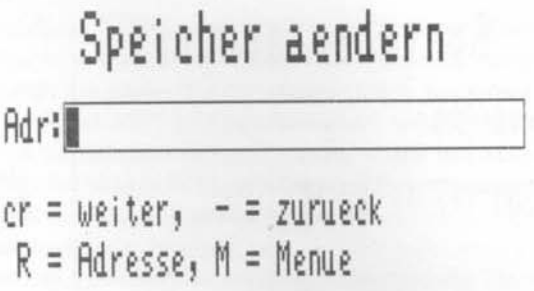


Abb. 2.1.5
Ändern von Speicherinhalten

Abb. 2.1.6
Eingabe einer neuen
Adresse mit „r"

Abb. 2.1.7
Der Speicherinhalt
ab Adresse $9800

```
Speicher aendern

Adr:$9800

009801:FF FF ED FF

%10100101

cr = weiter,  - = zurueck
 R = Adresse, M = Menue
```

Abb. 2.1.8
Eingabe einer Dualzahl

```
Speicher aendern

Adr:$9800

009800:00 01 FF ED

$123.w

cr = weiter,  - = zurueck
 R = Adresse, M = Menue
```

Abb. 2.1.9 (rechts)
Eingabe einer Sedezimalzahl

```
Speicher aendern

Adr:$9800

009800:00 00 5A 66

3*($55a/3-10).w

cr = weiter,  - = zurueck
 R = Adresse, M = Menue
```

Abb. 2.1.10
Die Eingabe von mathematischen Formeln

Übrigens dürfen zwischen den einzelnen Ziffern und Zeichen keine Leerstellen stehen. Leerstellen sind für andere Zwecke reserviert. Die Leerstelle kann man durch Drücken der langen Taste auf der Tastatur, die auch „Space" genannt wird, erreichen.

Jetzt bestätigen wir unsere Eingabe noch per „CR" und mit Ausnahme der Werte für die Speicherinhalte erhalten wir *Abb. 2.1.7* auf dem Bildschirm. Der Inhalt der Speicherzellen ist undefiniert und kann jeden beliebigen Wert annehmen, daß Speicherzellen beim Spannungseinschalten in einen beliebigen Zustand kippen.

Jetzt können wir unsere ersten Eingabeversuche machen.

Es soll der Wert 0 in die erste Speicherzelle geschrieben werden. Dazu wird die Taste 0 gedrückt. Zum Abschluß drückt man die Taste „CR", wie gehabt. Es erscheint jetzt der Inhalt der Speicherzelle $9801 an der ersten Stelle nach dem Doppelpunkt, zusätzlich wird der Inhalt der Speicherzelle $9805 am rechten Ende der Zeile sichtbar. Wir geben jetzt mal einen weiteren Wert ein. Diesmal eine binäre Zahl. Man muß bei binären Zahlenwerten das Prozentzeichen dem Zahlenwert voranstellen. Wenn man die Zahl 10100101 eingeben will, so schreibt man:

```
% 1 0 1 0 0 1 0 1
```

Abb. 2.1.8 zeigt die Eingabe; wie immer bestätigen wir per „CR", und es erscheint die nächste Speicherzelle: $9802.

Nun wollen wir überprüfen, ob die Eingabe auch angenommen wurde. Dazu kann man im Speicher schrittweise zurückgehen, oder einfach die Adresse neu eingeben.

Drücken wir die Taste „-" und dann „CR", so erscheint der Inhalt der Speicherzelle $9801 wieder auf dem Bildschirm. Wenn man nochmals die Tasten „-" und „CR" betätigt, so erhält man auch den Inhalt der Speicherzelle $9800. Es muß jetzt auf dem Bildschirm der Wert 00, gefolgt vom Wert A5, neben der Adresse 9800 stehen. Die beiden rechts daneben stehenden Speicherzellen sind aber noch immer undefiniert.

Unsere bisherigen Eingabeoperationen bezogen sich immer auf eine Eingabebreite von 8 Bit, also auf ein Byte. Dies ist auch die Größe einer Speicherzelle. Bei den Prozessoren 68000 und 68008 ist ein Wort 16 Bit lang. Manchmal ist es ganz nützlich, diese Größe direkt eingeben zu können.

Wenn man ein Wort in den Speicher einschreiben will, so muß man dem Zahlenwert einfach die Zeichenfolge ".w" anhängen. Das Grundprogramm weiß dann, wie es den Wert auf die Speicherzellen verteilen muß. Wir versuchen es einmal und geben als Beispiel die folgende Zeichenfolge ein:

```
$ 0 1 2 3 . w
```

Die *Abb. 2.1.9* zeigt uns, wie sich unser Menue verändert hat. Dann die Taste „CR" drücken, und das Grundprogramm hat zwei Bytes auf einmal abgelegt. Mit „-" und „CR" bitte wieder zur Adresse $9800 zurückgehen. Dort steht jetzt die Sequenz: 01 23.

Das erste Byte enthält demnach den höherwertigen Teil der Zahl und das nachfolgende Byte auf Adresse $9801 den niederwertigen Teil.

Bei der Eingabe der Beispielszahl $123 hätte man das „.w" auch weglassen können, denn das Grundprogramm erkennt die Tatsache, daß es sich um ein Wort und nicht um ein Byte handelt auch an der Länge der eingegebenen Zeichenkette. $123 läßt sich ja nicht in einem Byte unterbringen. Wenn man aber die Zahl „$23" als Wort ablegen will, so muß man ".w" anhängen, denn die Zahl belegt normalerweise nur ein Byte.

Langworte sind 32 Bit lang und kommen bei unseren Prozessoren ebenfalls vor. Auch sie kann man unmittelbar eingeben. Dazu muß man ".l" anhängen. Beispiel: Die Zahl $5a66 soll als Langwort abgelegt werden. Dazu gibt man ein:

$ 5 a 6 6 . l

Nach Eingabe von „CR" erscheint der Inhalt der Speicherzelle $9804, wenn man vorher auf $9800 war. Es wurden also vier Speicherzellen belegt. Wenn man sich den Inhalt der Speicherzellen auf $9800 danach ansieht, so wurden die Werte 00 00 5A 66 abgelegt. Diesmal war das ".l" auch wirklich nötig.

Wenn man nur ein Byte ablegen will und eine längere Zahl hat, so kann man das mit ".b" erreichen, doch von dieser Möglichkeit muß man erst beim Arbeiten mit Namen und Rechnungen Gebrauch machen.

Und da sind wir auch schon beim nächsten Punkt: Arithmetik.

Man kann auch arithmetische Ausdrücke eingeben. Alle vier Grundrechenarten und die Klammertechnik sind erlaubt.

Beispiel: Das Ergebnis des folgenden Ausdrucks soll als Wort in den Speicherzellen $9800 und $9801 abgelegt werden:

3 · ($55A/3-10)

Dazu gibt man einfach ein:

3 * ($ 5 5 A / 3 - 1 0) . w

Achtung: Es dürfen keine Leerzeichen zwischen den angegebenen Zeichen stehen, wenn man den Ausdruck eingibt. *Abb 2.1.10* zeigt die Eingabe. Nach Drücken der Taste „CR" wird das Ergebnis vom Grundprogramm ermittelt und der Wert gemäß der Angabe „.w" als Wort abgelegt. Das Ergebnis lautet $53A, in der Speicherzelle $9800 wird daher der Wert 5 und in $9801 der Wert 3A abgelegt. Die Division wird übrigens nur ganzzahlig ausgeführt, das heißt Nachkommastellen werden abgeschnitten.

Neben den Grundrechenarten kann unser Grundprogramm aber auch noch weitere Operationen ausführen. Da gibt es die sogenannte Modulo-Operation. Sie wird mit dem umgekehrten Divisionsstrich angegeben, der auf deutschen Tastaturen leider nur als „Ö" (großer Buchstabe, Achtung ggf. SHIFT verwenden) erreichbar ist. Die Modulo-Operation ermöglicht die Rechnung mit Restklassen. Beispiel:

10 / 2 ergibt 5 Rest 0

Die Operation 10 \ 2 liefert also das Ergebnis 0.

Beispiele:

17 / 4 ergibt 4 Rest 1

damit ergibt 17 \ 4 also den Wert 1.

17 / 3 ergibt 5 Rest 2

damit ergibt 17 \ 3 den Wert 2.

Die Restklassen sind für einige Anwendungen beim Programmieren sehr nützlich. *Abb. 2.1.11* zeigt ein Eingabebeispiel, das den Wert drei in Speicherzelle $9800 ablegt.

Die Eingabe von negativen Zahlen erfolgt einfach durch Voranstellen des Minus-Zeichens. Beispiel: Die Zahl $555 soll als negative Zahl eingegeben und als Langwort abgelegt werden. Dazu gibt man ein:

– $ 5 5 5 . l

Dann wird der Wert $FFFFFAAB im Speicher abgelegt. Die Darstellung erfolgt im Zweierkomplement.

Logische Verknüpfungen sind ebenfalls möglich.

Das Zeichen WELLE wird zur Einer-Komplementbildung, also der reinen Negation gebraucht, mit dem Ausrufezeichen kann man eine Oder-Verknüpfung erreichen, und mit dem Ampersand wird eine Und-Verknüpfung ausgeführt. *Abb. 2.1.12* zeigt ein Beispiel mit einer Eingabekombination. Das Ergebnis ist die Zahl 9, die als Wort abgelegt wird. Dabei ergibt die Oder-Verknüpfung von 5 mit 8 zunächst den Wert 13. Binär geschrieben sieht das so aus:

```
      0101
ODER 1000
ergibt: 1101
```

Dann wird das Zwischenergebnis mit dem Wert der Negation von 4 Und-verknüpft.

NICHT 4 ist binär:
1111 1111 1111 1011,
wenn man als Gesamtgröße ein Wort zu Grunde legt. Wird diese Zahl mit dem Wert 1101 UND-verknüpft, so erhält man:

```
      0000 0000 0000 1101
UND   1111 1111 1111 1011
ergibt: 0000 0000 0000 1001,
```

also dezimal den Wert 9.

Wenn man übrigens nur einfach den Speicherinhalt ansehen will, so kann man auch nur die Taste „CR" drücken. Dann bleibt der alte Inhalt erhalten, und es wird nur die Adresse um eins erhöht. Im unteren Menuefeld sieht man eine Zusammenstellung aller Tasteneingaben, wie sie zur Steuerung dieses Menues verwendet werden können.

Mit den weiteren Möglichkeiten bei der Zahleneingabe wollen wir uns später beschäftigen und kehren jetzt einmal zurück ins Grundmenue; dazu geben wir „M" und „CR" ein.

2 = starten

Der Menuepunkt „2 = starten" wird benötigt, um Programme zu starten. Wir rufen ihn einfach mal auf. Das Grundprogramm fordert uns zur Eingabe der Startadresse auf. Wir haben aber noch kein Programm eingegeben, und daher geben wir auch keine Startadresse ein, sondern drücken die Taste „CR", um in das Grundmenue zurückzugelangen.

3 = ansehen

Der Menuepunkt „Speicher ansehen" ist da schon interessanter. Auch hier wird wieder eine Anfangsadresse verlangt. Wir geben die Adresse $3800 ein, drücken also die Tasten:

$ 3 8 0 0

und bestätigen mit der „CR"-Taste.

Es erscheint *Abb. 2.1.13*. Besitzer von anderen Grundprogrammversionen als 4.3 werden ggf. ein anderes Bild auf dem Schirm bekommen. Bei Adresse $3800 handelt es sich um einen EPROM-Bereich, der natürlich von Version zu Version unterschiedlich ist.

Links stehen die sedezimal dargestellten Anfangsadressen des jeweiligen Speicherbereichs. In einer Zeile sind 16 Speicherzelleninhalte dargestellt. Insgesamt wird der Inhalt von 128 Bytes angezeigt. Man kann nun im Speicher blättern, indem man mit der Taste „+" vorwärts, oder mit der Taste „-" rückwärts schreitet. Geben wir einmal die Taste „+" ein. Der nächste Bereich erscheint auf dem Bildschirm, diesmal beginnend bei Adresse $3840; es wird also immer um 1/2 Bildseite geblättert. Die Eingabe von „CR" ist dabei unnötig, da ja auch kein Cursor erscheint.

Wenn man nun „-" drückt, so erscheint wieder das ursprüngliche Bild ab Adresse $3800. Neben den sedezimalen Werten der Speicherzellen findet man auch eine Textinterpretation darunter. Dabei wird zum Speicherinhalt das jeweilige Zeichen aus der ASCII-Tabelle ausgegeben. Damit ist es möglich, Texte, die im Speicher stehen, auch als solche zu erkennen.

Wenn man die Taste „R" drückt, kann man eine neue Adresse eingeben, und mit „M" gelangt man zurück ins Grundmenue.

4 = *Symbole*

Jetzt zu einer Sache, die am Anfang dieses Kapitels schon mal kurz erwähnt wurde. Es gibt die Möglichkeit, einem bestimmten Adreßwert ein Symbol zuzuweisen. Der Sinn dieser Geschichte liegt darin, daß wir uns Symbole leichter merken können als nichtssagende Zahlen. Diese Möglichkeit erleichtert also den Umgang mit unserem System erheblich.

Zunächst rufen wir den Menuepunkt „1 = aendern" auf, dann tippen wir folgende Buchstaben ein:

r a m s t a r t : = $ 9 8 0 0

Abb. 2.1.14 zeigt die Eingabe. Wenn man anschließend die Taste „CR" drückt, so verschwindet die Eingabe, und das leere Eingabefeld wird wieder sichtbar. Das ist ok so, denn wir wollten ja gar keine Adresse eingeben, sondern nur ein Symbol definieren. Das Symbol trägt die Bezeichnung „RAMSTART" und besitzt den Wert $9800.

Jetzt drückt man wieder die Taste „CR" und gelangt so ins Grundmenue zurück, ohne das Menue „aendern" weiter ausgeführt zu haben.

Nun die Taste „4" und „CR" drücken, um den Menuepunkt „4 = Symbole" auszuwählen. In der ersten Zeile ist unser definiertes Symbol zu sehen. Daneben steht der Wert im sedezimalen Zahlensystem und wiederum daneben das sogenannte Attribut. Es gibt an, welche Wortbreite das definierte Symbol besitzt. Die Ziffer „1" steht für Byte, „2" für Wort und „3" für ein Langwort. Mit „5" sind Symbole gekennzeichnet, die verwendet wurden, denen aber kein Wert zugewiesen wurde.

Immer wenn das Grundprogramm zur Eingabe einer Adresse auffordert, kann man auch eine Symboldefinition durchführen, ohne die Eingabe zu behindern. Damit ist es

```
Speicher aendern

Adr:$9800

009800:05 3A 5A 66

17\14
```

```
cr = weiter,  - = zurueck
 R = Adresse, M = Menue
```

Abb. 2.1.11 Die Modulo-Operation

```
Speicher aendern

Adr:$9800

009800:FF FF FA AB

¬4&(5!8).w
```

```
cr = weiter,  - = zurueck
 R = Adresse, M = Menue
```

Abb. 2.1.12
Eingabe mit logischen Verknüpfungen

```
Speicher ansehen

+=weiter -=rueckw R=Adr M=Menue

---- 0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
00003800 >61 00 EC 0E 60 E6 B0 3C 00 32 66 00 00 08 61 00
00003800 a     l     '  f  0  <     2  f              a
00003810 EC B4 60 D8 B0 3C 00 33 66 00 00 0A 61 00 E7 06
00003810 l  4  '  X  0  <     3  f              a     g
00003820 60 00 FC 2E B0 3C 00 57 66 C2 60 00 FC 24 49 4E
00003820 '     |  .  0  <     W  f  B  '     |  $  I  N
00003830 54 45 52 52 55 50 54 00 42 55 53 45 52 52 4F 52
00003830 T  E  R  R  U  P  T     B  U  S  E  R  R  O  R
00003840 00 41 44 52 45 52 52 4F 52 00 49 4C 4C 45 47 41
00003840    A  D  R  E  R  R  O  R     I  L  L  E  G  A
00003850 4C 20 49 4E 53 54 52 00 44 49 56 20 42 59 20 5A
00003850 L     I  N  S  T  R     D  I  V     B  Y     Z
00003860 45 52 4F 00 43 48 45 43 4B 00 54 52 41 50 56 00
00003860 E  R  O     C  H  E  C  K     T  R  A  P  V
00003870 50 52 49 56 49 4C 45 47 45 00 54 52 41 43 45 00
00003870 P  R  I  V  I  L  E  G  E     T  R  A  C  E
```

Abb. 2.1.13
So kann ein Speicherbereich aussehen

Abb. 2.1.14 Die Definition eines Namens

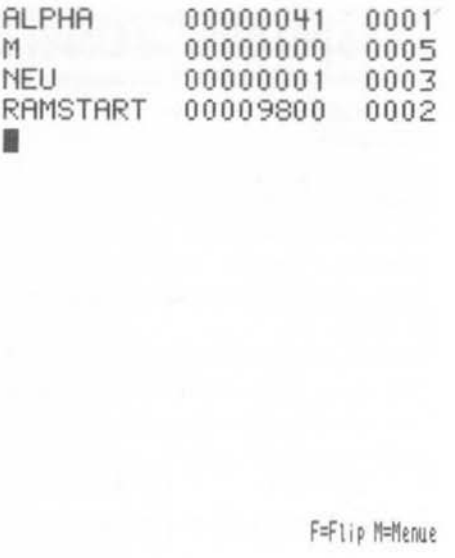


Abb. 2.1.15 Die Symboltabelle nach dem Eintrag von weiteren Namen

+	Addition 32 Bit
-	Subtraktion 32 Bit
*	Multiplikation 32 Bit ab 4.0
/	Division 32 Bit ab 4.0
\	Modulodivision ab 4.0
( )	Klammerrechnung
!	Oder-Verknüpfung ab 4.0
&	Und-Verknüpfung ab 4.0
*	Als Operand, Programmstand
?	Textanfangsadresse
'text'	Text. als ASCII-Wert
~	Nicht-Verknüpfung
-	Vorzeichen einer Zahl
$nnnnnnnn	sedezimale Zahl
ddddddddd	dezimale Zahl
%bbbbbbbb	binäre Zahl
@symbol	Adresse eines festen Symbols
!symbol	Index eines festen Symbols

Abb. 2.1.16
Arithmetische Operationen

möglich, sich schnell mal ein paar Namen mit häufig verwendeten Werten zu definieren, ohne jedesmal Zahlen eingeben zu müssen.

Wir wollen jetzt ein weiteres Symbol definieren. Dazu geht man zurück ins Grundmenue (Drücken der Taste „M") und ruft wieder „aendern" auf. Dann wird folgendes eingegeben:

```
neu:=1.l   CR
alpha:='A' CR
m          CR
m          CR
```

Die letzten beiden Eingaben sind etwas eigentümlich und bewirken folgendes: Mit dem ersten „m" und „CR" gelangt man nicht ins Grundmenue, sondern es wird die Adresse 0 angezeigt, und das Eingabefeld erscheint unten. Mit dem nächsten „m" und „CR" gelangt man wieder ins Grundmenue zurück.

Nun schauen wir uns die Symboltabelle an, um zu verstehen, was passiert ist. *Abb. 2.1.15* zeigt den Bildschirminhalt. Die Ausgabe der Symboltabelle erfolgt in alphabethischer Reihenfolge. Das Symbol Alpha besitzt den Wert $41, das ist der ASCII-Wert des Buchstabens „A". Dann folgt ein Symbol „M". Doch wo wurde das definiert? Nun, bei der Eingabe von „m" und „CR" im Änderungsmenue. Dort wurde versucht, ein Symbol aufzurufen, das noch nicht definiert war. Es wird automatisch in die Symboltabelle eingetragen, und der Wert ist 0. Das Attribut zeigt durch den Wert 5, daß es sich um ein undefiniertes Symbol handelt. Dann folgt das Symbol mit dem Namen „Neu", das den Wert 1 besitzt und das Attribut 3, denn es wurde mit ".l" als Langwort definiert. Schließlich folgt noch das Symbol „Ramstart", das wir bei unserem ersten Versuch definiert haben. In Ihrer Liste können sich auch noch weitere undefinierte Symbole befinden, wenn Sie bei den bisherigen Eingaben Tippfehler gemacht haben und dadurch unwillkürlich Symbole eingetragen haben. Die Symbole kann man auch wieder löschen; dazu gibt es ein eigenes Menue im Optionenmenue.

Da die Liste im Prinzip beliebig lang werden kann, gibt es eine Möglichkeit, den Ausgabevorgang anzuhalten. Dazu drückt man die Tasten CTRL und dann die Taste „S" dazu. Soll die Ausgabe fortgesetzt werden, so drückt man die Tasten CTRL und dann „Q" dazu.

Die Symboltabelle läßt sich auch auf einem Drucker ausgeben. Dazu verwendet man das Optionenmenue, wie später noch gezeigt wird.

Abb. 2.1.16 zeigt abschließend noch eine Zusammenstellung der Eingabemöglichkeiten von Werten. Zwei Möglichkeiten haben wir noch nicht besprochen, „@SYMBOL" und "!SYMBOL". Damit kann man fest im Grundprogramm eingebaute Unterprogramme ansprechen, oder genauer gesagt, im ersten Fall die Adresse des Unterprogramms ermitteln und im zweiten Fall den Index, für einen besonderen Aufruf. Wie das geht, wird ausführlich im Kapitel 3 besprochen.

2.1.2 Das Editormenue

Zunächst sei an dieser Stelle an die Abb. 2.1.4 erinnert, in der alle hier besprochenen Menues übersichtlich dargestellt sind. Schauen wir uns mal das zweite Menue an. Dort kann man den Texteditor aufrufen, aber auch den Assembler, sowie die Bibliotheksfunktion und das Optionenmenue. Den Editor wollen wir hier nur anhand eines kurzen Beispiels ansehen. Wegen seines Umfangs ist ihm der Abschnitt 2.2 gewidmet. Den Assembler, wie auch die Bibliotheksfunktionen, lernen wir in späteren Kapiteln anhand von Beispielen nach und nach kennen. Auch für diese Möglichkeiten nehmen wir uns vorerst nur je ein kleines Beispiel vor.

1=*Editor*

Also rufen wir den Editor auf (die Tasten „1" und „CR" drücken). Im unteren Bildrand erscheinen einige Zusatzangaben, die uns jetzt noch nicht weiter interessieren. Die Karofelder auf der linken Seite geben an, daß keine gültige Textzeile in der betreffenden Zeile vorhanden ist. Ganz oben links blinkt der Cursor. Der Texteditor ist also zur Eingabe eines Textes bereit. Wir tippen einmal einen Buchstaben ein, z. B. „S". Er erscheint links oben im Bildfenster, der Cursor ist dann um eine Stelle nach rechts gewandert. Wenn man nun einen weiteren Buchstaben eingibt, so erscheint er dahinter, und der Cursor wandert wieder weiter. So kann man wie auf einer Schreibmaschine Texte eingeben. Hat man eine Zeile beendet, so drückt man die Taste „CR" und gelangt so in die nächste Zeile. Tippfehler lassen sich mit der Taste DEL beseitigen. Sie radiert den letzten Buchstaben wieder aus, der Cursor wandert dabei um ein Zeichen nach links zurück.

So, bitte jetzt einfach den Text in *Abb. 2.1.17* eingeben, so daß der Bildschirm genauso aussieht, wie dargestellt.

Wichtig ist, daß die Karozeichen weggelöscht bzw. überschrieben werden. Steht hinter dem Karozeichen ein Text, so wird er später vom Texteditor wieder entfernt.

Das Eingetippte ist übrigens ein einfaches Assemblerprogramm, das uns hier aber nur als Beispiel dient. Der Editor besitzt eine Vielzahl von Funktionen zur Textkorrektur und -Bearbeitung, dazu aber mehr in Abschnitt 2.2. Nun soll der Texteditor wieder verlassen werden. Dazu muß man folgende Tastenfolge drücken. Zuerst die Tasten „CTRL" und „K" gleichzeitig. Dann die Taste „X" drücken. Achtung, bei älterern Grundprogrammversionen mußte man beim „X" einen Großbuchstaben eingeben, bei der neuen Version ist das nicht mehr nötig.

Man befindet sich jetzt wieder im Editormenue. Jetzt wird einmal der Assembler, also die Funktion 2 aufgerufen.

```
start:
 move #100,d0
 jsr @schreite
 rts
```

Textstart=009000 Fenster=009000 Tor=009000 amer CTRL-J=Hilfe

Abb. 2.1.17
Ein erster Text zur Eingabe

```
Rolf-D.Klein 68000/08 Assembler 4.3 (C) 19E

009C00                      START:
009C00   303C 0064           MOVE #100,D0
009C04   4EB9 00001288       JSR @SCHREITE
009C0A   4E75                RTS
009C0C

008AA8   Ende-Symboltabelle          F=Flip M=Menue
```

Abb. 2.1.18
Nach dem Aufruf des Assemblers

```
Bibliothek

Name   : UFORM68K
Start  : 00C99A
Laenge : 0016F6

Starten J = JA
cr = weiter, M = Menue
```

Abb. 2.1.19
Beispiel einer Bibliothek

2=Assembler

Der Assembler wurde jetzt automatisch durch das Aufrufen der Menuepunkte gestartet. Er assembliert das soeben mit dem Texteditor eingegebene Programm. *Abb. 2.1.18* zeigt das Ergebnis. Ganz links sind die Speicheradressen angegeben, in der mittleren Spalte ist der Maschinencode gezeigt, der durch den Assembler erzeugt wurde, und rechts befindet sich das Quellprogramm, so wie es mit dem Texteditor eingegeben wurde. Unten ist noch eine Angabe zur Symboltabelle enthalten. Dieser Wert kann unterschiedlich sein, je nachdem, ob man die Versuche des ersten Abschnitts durchgeführt hat, oder nicht. Diese Adresse gibt die erste freie Speicherzelle nach der Symboltabelle an. Man muß aufpassen, daß dieser Wert nicht zu groß wird. Wenn man sehr lange Programme schreibt, kann das passieren. Dann würde die Symboltabelle in den Textbereich ragen und ggf. Teile zerstören. Der Assembler würde diesen Fall ebenfalls erkennen und auf dem Bildschirm groß die Meldung „ADRESS ERROR" ausgeben. Als Abhilfe wären dann alle Symbole zu löschen und schlimmstenfalls das Programm, mit einer neuen Anfangsadresse (siehe Optionenmenue), neu einzugeben. Der für den Texteditor voreingestellte Speicherbereich beginnt bei Adresse $9000.

Nun zurück in das „Editormenue". Als nächsten Punkt nehmen wir uns die Funktion „3=Bibliothek" vor.

3=Bibliothek

Zunächst ein paar Worte zur Aufgabe dieser Funktion. Unter einer Bibliothek versteht man eine Ansammlung von Programmteilen, die wir uns für spätere Anwendungen aufbewahren wollen. Im wesentlichen handelt es sich dabei um Programmteile, die öfter mal gebraucht werden und darum nicht jedesmal neu geschrieben werden sollen. Diese Bibliothek kann bei unserem System in einem EPROM oder auch im RAM-Bereich liegen.

Wir wählen die Bibliothek durch Eingabe von „3" an. Falls nun keine Bibliothek-EPROMs eingesetzt sind, so wird ein leeres Fenster erscheinen. *Abb. 2.1.19* zeigt ein Beispiel mit einem eingesetzten EPROM, das den Formatierer für Floppy-Laufwerke enthält. Dieser Formatierer hat den Namen „UFORM68K". Man kann nun eine solche Bibliothek aufrufen, indem man einfach die Taste „J" drückt; „J" steht für JA. Wenn man die Bibliothek nicht aufrufen will, sondern die nächste betrachten will, so gibt man einfach die Taste „CR" ein. Werden keine Bibliotheken mehr gefunden, so wird in der unteren Bildhälfte „F=Flip M=Menue" ausgegeben. Man kann dann die Bibliothek durch Eingabe des Buchstabens „M" verlassen und gelangt wieder ins Grundprogramm zurück.

Es können mehrere Bibliotheken im Speicher stehen. Um den Anfang einer Bibliothek zu finden, sucht das Grundprogramm im gesamten Hauptspeicher nach einem bestimmten Bitmuster. Findet es das Bitmuster, so wird die Information, wie Name, Startadresse etc. ausgegeben. Nun kann es vorkommen, daß zufällig einmal das notwendige Muster im Speicher steht, aber gar nicht zu einer Bibliothek gehört. In diesem Fall

würde eine fehlerhafte Anzeige im Fenster erscheinen. Da das Bitmuster sehr lang ist, ist die Wahrscheinlichkeit dafür aber gering.

Wie man selbst Bibliotheken erstellt, wird in einem späteren Kapitel anhand von Beispielen noch erklärt. Dazu müssen wir erst einmal in Assmbler programmieren lernen.

Nun zum letzten Menuepunkt des „Editormenues", dem Optionenmenue.

4=*Optionen*

Zunächst wählen wir die Funktion einfach einmal an. Es erscheint ein weiteres Menue mit vier Punkten.

Wir wählen als erstes den Menuepunkt „1=Assembler-Opt". „Opt" steht für Optionen, was soviel wie Erweiterungen oder Zusatz bedeutet. Nun kann man wiederum einen von vier Menuepunkten anwählen. Mit dem ersten Menuepunkt kann man die normale Textausgabe des Assemblers ausschalten, es erscheinen dann nur noch etwaige Fehlermeldungen auf dem Bildschirm und nicht mehr der erzeugte Maschinencode mit dem Quelltext. Eine Übersetzung unseres Beispielprogramms von vorhin würde dann wie *Abb. 2.1.20* aussehen. In der zweiten Zeile wird die Adresse der ersten freien Speicherzelle nach dem abgelegten Maschinencode angezeigt ($9C0C), in der dritten steht das Ende der Symboltabelle. Die Ausgabe ohne Quelltext geht viel schneller, darum sollte man sie immer anwenden, wenn man sich nicht für den Maschinencode interessiert.

Alle weiteren Funktionen in diesem Untermenue beziehen sich ebenfalls auf die Ausgabe während des Assembliervorganges. Die zweite Funktion „2=Ausgabe auf CRT" ist voreingestellt. Das bedeutet, die Ausgabe erfolgt auf dem Bildschirm. Wurde durch den Aufruf von „1" die Ausgabe abgeschaltet, so kann man sie durch „2" wieder einschalten. Der dritte Menuepunkt leitet die Ausgabe auf den Drucker um (siehe Kapitel mit dem Abschnitt „Der Druckeranschluß„).

Manche Drucker führen beim Erhalt des Zeichens CR (carriage return) auch gleich eine Zeilenschaltung durch. Schickt nun der Computer, wie normalerweise üblich, das Zeichen LF (line feed) hinterher, so entsteht im Ausdruck nach jeder Zeile eine Leerzeile. Um dies zu verhindern, ist es mit der Funktion „4" möglich, die Ausgabe wie bei „3" auf den Drucker umzuleiten, jedoch ohne einem „LF„-Zeichen am Zeilenende.

Mit der Funktion „2=Textstart" ist es möglich, die Startadresse des Textgebietes zu verändern. Diese Änderung gilt für den Arbeitsbereich des Editors genauso, wie für die Einstellung des Assemblers oder anderer Übersetzungsprogramme. Der zu übersetzende Quellcode wird automatisch von dieser neuen Stelle geladen. Wir rufen zunächst die Funktion „2" auf und wählen dann „2=neuer Text". Es erscheint *Abb. 2.1.21*. Wir werden zur Eingabe der neuen Startadresse aufgefordert. Wählen wir die Adresse $9800. Im Prinzip kann man jede beliebige Adresse eingeben, sie muß jedoch im RAM liegen und darf nicht mit anderen Bereichen, wie zum Beispiel der Symboltabelle oder dem Arbeitsspeicher des Grundprogramms kollidieren. Mit „CR" wird die Eingabe abgeschlossen, und es erscheint *Abb. 2.1.22*. Das Grundprogramm fragt, ob man es

```
Rolf-D.Klein 68000/08 Assembler 4.3 (C) 19E

009C0C

008AA8  Ende-Symboltabelle
```

F=Flip M=Menue

Abb. 2.1.20
Aufruf des Assemblers, mit „nur Fehlerausgabe"

```
Text neu

Adr:
```

Abb. 2.1.21
Angabe der Startadresse für einen neuen Text

```
Text neu

Adr:$9800
```

```
start:
 move #100,d0
 jsr @schreite
 rts
```

Textstart=009000 Fenster=009000 Tor=009000 amer CTRL-J=Hilfe

Abb. 2.1.23 Der Texteditor mit 80 Zeichen pro Zeile

```
wirklich ?, Ja=J
```

Abb. 2.1.22
„J" eingeben, wenn die neue Adresse verwendet werden soll

```
Rolf-D.Klein 68000/08 Assembler 4.3 (C) 1984, Seite 1

009C0C

008A60  Ende-Symboltabelle
009048  Ende-Debug-Tabelle
```

F=Flip M=Menue

Abb. 2.1.24 Wenn „Debug" im Optionen-Menue eingeschaltet wurde, dann gibt der Assembler neben dem Symboltabellen-Ende auch das Ende der Debug-Tabelle aus

wirklich ernst meint, einen neuen Textbereich anlegen zu wollen, denn ein alter Text der möglicherweise auf dieser Adresse liegt, wird dadurch zerstört. Wenn man die Taste „J" drückt, so gelangt man ins Grundmenue zurück, und der Textbereich wurde angenommen. Betätigt man eine andere Taste, so wird die Adresseingabe ignoriert. Wenn wir nach der Adressänderung den Texteditor aufrufen, so ist das Textfenster leer. Im unteren Bildschirmteil findet man die Angabe „Textstart=009800", die den neuen Textstart angibt. Nun wieder zurück ins Optionenmenue; den Editor verläßt man mit „CTRL und K" und dann „X".

Jetzt haben wir in dieser Reihe von Möglichkeiten noch den Menuepunkt „1=alter Text" auszutesten. Rufen wir ihn auf, so verlangt das Grundprogramm ebenfalls eine Adresse. Diesmal geben wir die Adresse $9000 ein. Dort befindet sich der Text, den wir früher als Beispiel für den Editor eingegeben haben. Diesmal erscheint nicht die Frage „wirklich ? ja=j", denn man kann dabei einen bestehenden Text nicht zerstören. Wenn man schließlich nochmals in den Editor geht, so findet man den ursprünglich eingegebenen Text wieder.

Nun wieder ins Optionenmenue zurück. Dort wählt man die Funktion „3=Loeschen Symbole" an. Sollten die Symbole tatsächlich gelöscht werden, wie wir es jetzt nicht tun wollen, so wäre an dieser Stelle die Folge „1" „CR" einzugeben. Ein Löschen der gesamten Symboltabelle, wie es diese Funktion bewirkt, ist nur notwendig, wenn die Symboltabelle z. B. durch das Anlegen zu vieler Symbole zu groß geworden ist, oder wenn sie mit dem Textbereich des Editors kollidierte (ADRESS ERROR).

Der nächste Menuepunkt im Optionenmenue „4=Zeichen/Zeile & Debug" bietet vier Unterpunkte.

Nach dem Einschalten der Spannung ist der Bildschirm auf eine Darstellung von 40 Zeichen pro Zeile eingestellt. Von dieser Einstellung betroffen sind z. B. der Editor, der Assembler und die Symbolausgabe, sowie alle höheren Sprachen und Compiler, die z. B. über die Bibliothek angewählt werden, sofern sie nicht selbst diesen Wert anders einstellen.

Wie aus dem Untermenue nach dem Aufruf der Funktion 4 zu erkennen ist, kann das System durch die Eingabe einer „2" auf 80 Zeichen/Zeile umgestellt werden. Eine „1" an dieser Stelle stellt wieder auf den Standardwert zurück. Wir schalten einmal die Ausgabe auf 80 Zeichen pro Zeile um und drücken dazu die Taste „2". Dann geht man in den Editor und es zeigt sich *Abb. 2.1.23*, falls der alte Text noch vorhanden ist. Damit kann man den Editor jetzt auch für Textverarbeitung einsetzen. Übrigens werden im Mode „40 Zeichen/Zeile" alle Zeichen, die nach der 40ten Stelle liegen, einfach nicht dargestellt, man sieht gewissermaßen nur einen Ausschnitt.

Die letzten beiden Punkte sind für den Einzelschrittbetrieb bestimmt. Wenn man durch den Aufruf von „3" den sogenannten Debug-Mode einstellt, so werden später beim Einzelschritt die Quellzeilen neben der aktuellen Adresse sichtbar, und man tut sich leichter bei der Fehlersuche. Der Assembler konstruiert eine sogenannte Debug-Tabelle und plaziert sie hinter dem abgelegten Text. In der Assemblermeldung erscheint dann zusätzlich die Endadresse der Debug-Tabelle, wie *Abb. 2.1.24* zeigt.

Da die Debug-Tabelle viel Platz braucht, sollte man sie wieder ausschalten, wenn man sie nicht benötigt.

2.1.3 *Das Speichermenue*

In diesem Menue befinden sich die Befehle zum Speichern und Laden von Programm und Daten. Schauen wir uns zunächst die Funktionen für das Arbeiten mit einem Cassettenrecorder an.

1=Speichern CAS

Als erstes rufen wir einmal „1=Speichern CAS" auf. Man gelangt in ein weiteres Menue und muß auswählen, ob man Texte oder Daten speichern will. Wenn man den Editorinhalt abspeichern will, so muß man „1" eingeben; will man aber Programme oder beliebige Daten im Maschinencode abspeichern, , so gibt man „2" ein.

Zunächst zu den Texten. Wir drücken die Tasten „1" und „CR". Der Computer fragt uns nach der Startadresse des Textes. In unserem Beispiel geben wir $9000 (und wie immer „CR") ein. Wenn man sich die Adresse nicht merken will, so kann man auch einfach das Zeichen "?" eintippen, dann wird automatisch der aktuelle Textbereich verwendet, den auch der Editor beim nächsten Aufruf verwenden würde.

Das Grundprogramm will jetzt von uns einen Namen wissen (*Abb. 2.1.25*). Der hier einzugebende beliebige Text dient als Kennzeichnung für die abzuspeichernden Daten. Unser mit dem Editor eingegebenes Programm erhält also einen Namen. Wir geben mal „Hallo geht" ein. Dieser Name wird später beim Einlesen als Kontrolle wieder ausgegeben. Nun schaltet man den Cassettenrecorder auf Aufnahme und startet ihn. Dann wird die Taste „CR" gedrückt und der Text auf Cassette abgespeichert. Achtung, die Baugruppe CAS muß zuvor abgeglichen worden sein, siehe Kapitel 7. Danach meldet sich das Grundprogramm wieder.

Nun wollen wir einmal Daten abspeichern. Dazu wählt man das Menue „1=Speichern CAS" im „Speichermenue" und „2=Daten" an. Nun wird neben der Startadresse auch eine Endadresse angefordert. Beim Speichern von Texten war das nicht nötig, denn das Ende eines Textes wird vom Grundprogramm automatisch erkannt. Bei Daten sieht das anders aus, denn das Grundprogramm kann ja nicht wissen, welchen Bereich wir abspeichern wollen.

Also bitte eingeben: $9000 als Startadresse, $97FF als Endadresse. Als Name wählen wir „TEST", und vor dem Bestätigen des Namens per „CR" sollte unser Bildschirm wie *Abb. 2.1.26* aussehen. Bei der Namensangabe kann man beliebige Buchstaben verwenden. Dann den Cassettenrecorder wieder auf Aufnahme schalten und „CR" drücken. Nach einer Weile sind die Daten abgespeichert. Die Übertragung geschieht dabei mit 1200 Baud, das bedeutet ca. 100 Bytes pro Sekunde. Bei einem Speicherbereich von $9000 bis $97FF sind 2048 Bytes abzuspeichern, also muß man ca. 20 Sekunden warten, bis der Vorgang beendet ist.

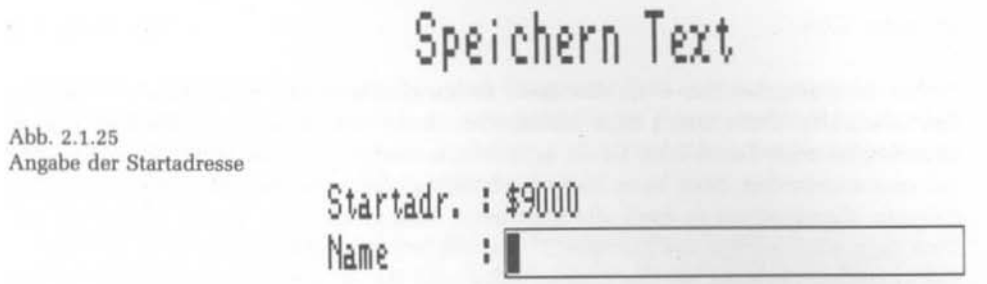


Abb. 2.1.25
Angabe der Startadresse

Abb. 2.1.26
Auch Namen können angegeben werden

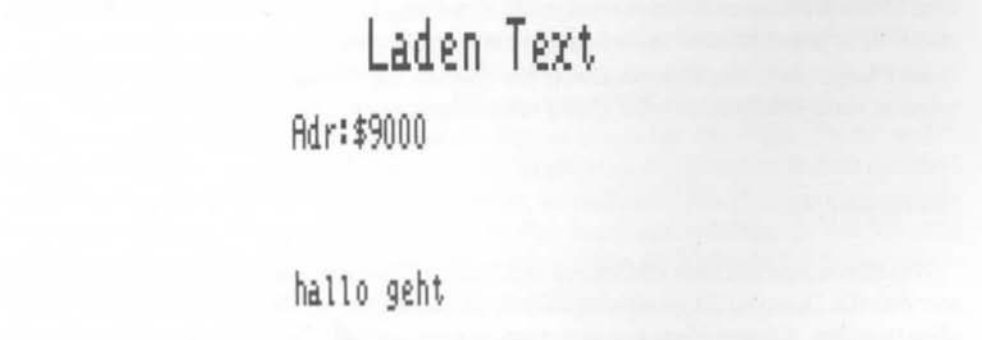


Abb. 2.1.27
Nach dem erfolgreichen Laden eines Textes

2=Laden CAS

Nun zum umgekehrten Weg; die Daten sollen also vom Cassettenrecorder zurück in den Computer. Dazu wählt man das Menue „2=Laden CAS". Dort muß man erneut angeben, ob man Texte oder Daten zurücklesen will. Außerdem wird auch hier eine Adresse angefordert. Man kann hier eine beliebige Adresse angeben, die nicht mit der Adresse übereinstimmen muß, die man beim Abspeichern des Textes angegeben hat. Man kann auch wieder das Zeichen „?" als Abkürzung angeben, wenn man den Text in den aktuellen Textspeicher des Editors laden will. Wenn sich dort aber ein anderer Text befindet, so wird er überschrieben. Die Cassette muß nun zunächst zurückgespult werden, dann wird die Startadresse eingegeben, also z. B. $9000, oder das Fragezeichen. Dann „CR" drücken und danach den Rekorder auf Wiedergabe schalten. Nun muß nach kurzer Zeit der Programmname auf dem Bildschirm erscheinen. Tut er das nicht, so muß man entweder die CAS-Baugruppe noch abgleichen (siehe Kapitel 7), oder man hat den Polaritätschalter in der falschen Stellung. Die richtige Lage muß man leider ausprobieren. Außerdem sollte der Aufnahmeschalter auf „AUS" stehen, um die Wiedergabe nicht zu beeinflussen.

Wenn ein Fehler erkannt wird, so erscheint die Meldung „Prüfsumme falsch" auf dem Bildschirm. Auch dann sollte man die Inbetriebnahme der Schaltung gemäß Kapitel 7 durchführen.

Wenn das Programm richtig geladen wurde, so erscheint *Abb. 2.1.27*. Das „OK" links unten gibt an, daß der Text ordnungsgemäß geladen wurde.

Nun probieren wir das Laden von Daten. Dazu wird der Menuepunkt „2=Laden CAS" und dann „2=Daten" angewählt. Diesmal erscheint nur der Text „Laden Daten" auf dem Bildschirm, eine Adresse wird nicht abgefragt, da Daten immer nur wieder an die ursprünglichen Adressen zurückgeladen werden können.

Jetzt kann man den Cassettenrecorder starten. Nachdem die Daten richtig geladen worden sind, erscheint wieder eine Fertigmeldung.

3=Pruefen CAS

Was hat es nun mit dem Prüfen auf sich? Es funktioniert genausowie das Lademenue, nur daß die Daten nicht wirklich geladen, sondern nur mit dem Speicherinhalt verglichen werden. Dieses Menue sollte man immer unmittelbar nach dem Abspeichern verwenden, um zu kontrollieren, ob die Daten oder Texte auch wirklich richtig abgespeichert wurden. Im Fehlerfall erscheint entweder die Meldung „Vergleichsfehler" oder „Prüfsummenfehler", je nach Art. Der Vergleichsfehler kann auch auftreten, wenn man den Speicherbereich zwischendurch verändert hat. Also die Prüffunktion wirklich nur unmittelbar nach dem Abspeichern anwenden. Speichert man z. B. RAM-Bereiche ab, die vom Grundprogramm selbst verwendet werden, also den Arbeitsspeicher ab Adresse $8000 oder am Ende des Speichers den sogenannten Stack, so kann es beim Prüfen natürlich auch zu einem Vergleichsfehler kommen, denn der Inhalt des Arbeitsspeichers ändert sich ständig.

4=Floppy Start

Damit ist es möglich, eine angeschlossene Floppy als Speicher zu verwenden (siehe Kapitel 6). Nach dem Aufruf wird der erste Datenblock der Floppy in den RAM-Bereich ab Adresse $9800 geladen und gestartet. Es hängt also vom Inhalt der Floppy ab, was nach dem Aufruf des Menuepunkts weiter passiert.

Achtung! Wenn man das Mikrodos, wie es im Kapitel 6 beschrieben ist, verwendet, so sollte man Texte erst ab Adresse $A000 ablegen, da der Bereich $9000 bis $9FFF vom Mikrodos benutzt wird. Soviel vorab, der Vollständigkeit des Menues wegen, zur Floppy. Hier wollen wir uns aber noch nicht weiter damit befassen, vorerst genügt uns der Cassettenrecorder. Wer unbedingt schon mit der Floppy arbeiten will, muß hier einen Ausflug nach Kapitel 6 unternehmen und dieses jetzt zuerst durcharbeiten.

2.1.4 Das EPROM-Menue

Wenn man Besitzer eines EPROM-Programmierers (Baugruppe PROMMER) ist, so kann man damit eigene Programme in EPROMs ablegen oder Programme, die sich in einem EPROM befinden, einlesen.

Rufen wir zunächst die Funktion „1=EPROM prog." auf. Das Grundprogramm verlangt anschließend nacheinander die Eingabe von drei Adressen. Die „von"-Adresse gibt an, von wo die Daten geholt werden sollen. Die „bis"-Adresse gibt die letzte zu programmierende Adresse an. Dann muß man noch den Wert der Adresse „nach" eingeben. Diese Adresse bestimmt die Zieladresse im EPROM. Normalerweise gibt man hier den Wert 0 an, da man die Daten von Anfang an in das EPROM legen will. Wenn man aber mehrere Datensätze in einem EPROM hintereinander ablegen will, so kann man dies durch eine entsprechende Angabe des „nach"-Wertes erreichen.

Wenn man diese drei Werte eingegeben hat, so erscheint die Frage „Bereit = B". Durch die Eingabe von „B" wird also der Programmiervorgang gestartet. Rechts daneben wird die aktuelle EPROM-Adresse ausgegeben, so daß man den Programmiervorgang kontrollieren kann. Dies ist besonders bei großen Bereichen wichtig, da der Vorgang sehr lange dauern kann. Pro Byte benötigt unser System 60 Millisekunden, wobei die eigentliche Programmierzeit 50 Millisekunden beträgt.

Nach erfolgreicher Programmierung erscheint im unteren Bildteil die Meldung „EPROM OK" (*Abb. 2.1.28*). Wenn das EPROM nicht richtig programmiert wurde, so wird eine Fehlermeldung „EPROM FEHLER" ausgegeben. Danach gelangt man durch Drücken der Taste „M" wieder ins Grundmenue zurück.

Wenn der Programmiervorgang nicht klappt, so gibt es mehrerer Fehlermöglichkeiten dafür. Zum einen kann das verwendete EPROM defekt oder auch nur nicht gelöscht sein. Zum anderen kann aber auch das Monoflop nicht richtig abgeglichen sein. Siehe auch Kapitel 7. Ferner muß man den richtigen Programmierstecker in die Fassung eingesetzt haben, der je nach EPROM-Typ unterschiedlich ist.

2=EPROM lesen

Mit der Funktion „2=EPROM lesen", kann man den Inhalt von EPROMs in einen Speicherbereich bringen.

Rufen wir die Funktion auf, so werden wir auch hier wieder nach drei Adressen gefragt. Diesmal ist aber die Bedeutung der Adressen etwas anders. Die „von"-Adresse ist die Adresse im EPROM. Will man den gesamten Inhalt eines EPROMs einlesen, so muß man hier den Wert 0 eingeben. Die „bis"-Adresse ist die letzte zu lesende Adresse im EPROM. Bei einem 2764-Baustein mit 8Kx8 Speicherzellen ist der Wert $1FFF, denn der Speicher besitzt 8192 Speicherzellen, und die letzte Speicherzelle hat die dezimale Adresse 8191. Den dezimalen Wert kann man natürlich auch direkt eingeben. Die „bis"-Adresse gibt die Speicheradresse an, wohin der Inhalt des EPROMs geladen werden soll, also z. B. $9800. Dort muß natürlich RAM vorhanden sein. Man muß hier aufpassen, daß man keinen Arbeitsspeicher des Grundprogramms oder gar eigene Texte und Programme überschreibt.

3=Speicherbereiche

Diesen Menuepunkt haben wir schon kennengelernt; er ist in Abb. 2.1.2 und Abb. 2.1.3 abgedruckt. Damit kann man sich über RAM-Bereiche informieren. Um festzustellen, in welchen Adressenbereichen sich RAM befindet, verwendet das System einen groben Speichertest. Theoretisch gibt es die Möglichkeit, daß RAM angezeigt wird, obwohl keine Bausteine eingesteckt sind. Das kommt allerdings ganz selten vor. Sollte dieser Fall eintreten, so ist dies kein Grund zur Beunruhigung. Der Grund liegt darin, daß die Busleitungen während des Ansprechens der nicht belegten Bereiche undefiniert sind. Nehmen sie nun zufällig einen Wert an, der das System vermuten läßt, hier liegt RAM vor, so kommt es zu dem beschriebenen Phänomen. Dieser Fehler kann ganz leicht vermieden werden, wenn man z.B. auf eine der Datenbusleitungen einen Widerstand nach +5 V legt, dann findet der Speichertest an diesem Bit garantiert immer eine 1, egal ob er 0 oder 1 einschreibt, und damit wird dann sicher zwischen RAM und unbelegtem Platz unterschieden. Diese Maßnahme ist aber nur bei ganz ungünstigen Fällen nötig, sie ist bei normalen Konfigurationen des Computers nicht notwendig. Die Angabe „Erster Bereich" ist aber immer exakt, denn dafür wird ein komplizierterer Speichertest verwendet. Wenn aber umgekehrt kein RAM angezeigt wird, obwohl eines da sein müßte, so liegt der Fehler woanders, z. B. an einer fehlerhaften Karte. Ebenfalls darf in EPROM-Bereichen kein RAM-Bereich angezeigt werden.

4=Text drucken

Mit dem letzten Menuepunkt im EPROM-Menue „4=Text drucken", wird der aktuelle Text, wie er zuletzt durch „Textstart" im Optionenmenue oder nach dem Einschal-

```
EPROM prog.

von      0
bis      $1fff
nach     0

Bereit = B   00001FFF

EPROM OK, M=Menue
```

Abb. 2.1.28
Nach dem Programmieren eines 8 K × 8-EPROMs

```
IO lesen                                F=Flip M=Menue

Adr:$ffffff68

        8D

    10001101

R=Adr D=Dauer S=Stop M=Menue
```

Abb. 2.1.29
Inhalt des Ports $FFFFFF68

```
IO setzen

Adr:$ffffff70

Data:
```

Abb. 2.1.30
Graphik-Prozessor-Port beschreiben

ten auf Adresse $9000 liegt, auf den Drucker ausgegeben. Dabei erfolgt hier die Druckausgabe immer mit LF (siehe Optionenmenue).

Diesen Menuepunkt kann man z. B. für Textverarbeitung verwenden, um beispielsweise Briefe auszudrucken, oder für höhere Programmiersprachen, um einfach mal nur die Quelle (also das Programm, wie man es eingetippt hat) auf Papier zu bekommen.

2.1.5 Das IO-Menue

1 = IO lesen

Wir wählen zunächst den Menuepunkt „IO lesen". Das Grundporgramm erwartet die Eingabe einer Adresse. Wir geben mal die Adresse $FFFFFF68 ein und schließen mit „CR" ab. *Abb. 2.1.29* wird sichtbar. Unter der Adresse findet man den sedezimalen Code: $8D, gleich darunter die binäre Darstellung: %10001101. Im unteren Bildteil werden uns die Eingabemöglichkeiten aufgezählt, die uns jetzt zur Verfügung stehen. Nun tippen wir mal den Buchstaben „A". Auf dem Bildschirm erscheint ein neuer Code. Die Adresse $FFFFFF68 ist nämlich die Adresse des Tastatur-Ports, also des IO-Bausteins, über den der Tastaturcode in den Rechner gelangt. Mit „IO lesen" kann man sich die Inhalte beliebiger Ports ansehen. Wenn man die Taste „R" drückt, so kann man eine neue Portadresse angeben, wenn man „D" drückt, so wird der Portinhalt andauernd eingelesen, und man kann dann Veränderungen des Wertes besser beobachten. Mit „S" stoppt man die Dauerfunktion. Durch die Eingabe von „M" gelangt man ins Menue zurück. Übrigens kann man sich damit natürlich auch Speicherinhalte ansehen, allerdings immer nur 1 Byte.

2=IO setzen

Mit dem Menuepunkt „IO setzen" ist der umgekehrte Vorgang möglich. Man kann dabei einen Wert an einen Port ausgeben. Dazu wählen wir das Menue an.

Das Grundprogramm verlangt wieder die Eingabe einer Adresse. Wir geben mal die Adresse $FFFFFF70 an. Es ist dies der Befehlsport des Graphikprozessors. Dann erscheint die *Abb. 2.1.30*. Es wird nach dem Datenwert verlangt. Man kann hier aber nur eine Größe angeben, die maximal 1 Byte umfaßt.

Wir geben nun den Wert $C ein, bitte nicht erschrecken, der Bildschirm wird dann ganz hell. Der Wert $C, hat den Graphik-Prozessor veranlaßt, den Bildschirm zu füllen. Unten links wurde dann vom Grundprogramm noch der Text „M=Menue R=Adr" eingefügt. Wenn man jetzt „R" eingibt, so kann man eine neue Adresse eingeben oder mit „M" das Menue verlassen. Das Menue ist sehr praktisch, wenn man Peripherie testen will. Wir haben es hier allerdings etwas mißbraucht.

Die vielen „F"s sind wichtig, da alle Ports bei unserem 68008-Computer auf der Adresse $FFFFFF00 beginnen und bei $FFFFFFFF aufhören. Bei der 68000-Version reicht der Bereich von $FFFFFE00 bis $FFFFFFFF, ist also doppelt so groß. Im Prinzip

könnte man einen noch größeren Bereich ansprechen, jedoch besitzen die meisten IO-Baugruppen nur eine Dekodiermöglichkeit für 8 Adreßleitungen, und das ergibt beim 68008 256 mögliche Adressen.

3=Einzelschritt

Nun zum letzten Menuepunkt, dem Einzelschritt. Wir wählen dazu „3=Einzelschritt" an. Das Grundprogramm fragt wieder eine Startadresse ab (*Abb. 2.1.31*). An dieser Stelle kann der Wert einer Adresse ebenso eingegeben werden, wie stellvertretend ein definiertes Symbol. Wir geben das Symbol START ein, falls das Programm, das wir am Anfang des Kapitels eingegeben haben, noch im Speicher steht. Falls es nicht mehr vorhanden ist, also der Computer in der Zwischenzeit mal abgeschaltet war, so kann man es mit dem Editor nochmal eingeben und dann mit dem Assembler übersetzen.

Abb. 2.1.32 zeigt den Bildschirm, nachdem das Symbol eingegeben wurde. Wie wir bereits vermuten können, enthält diese Bildschirmmeldung viele Informationen. Im Moment wollen wir uns auf die Bedienung des Grundprogramms beschränken. Die Einzelheiten dieser Informationen werden uns nach und nach im Kapitel 3 vertraut.

Wenn man nun die Taste „CR" zweimal drückt und außerdem das im Speicher liegende Programm richtig eingegeben sowie fehlerfrei per Assembler übersetzt hat, so erscheint die *Abb. 2.1.33*. Das Programm wurde also nicht in Einzelschritten ausgeführt, sondern lief in einem Zug durch. Eine Linie wurde gezeichnet. Betätigen wir jetzt ein weiteres Mal die „CR„-Taste, so sind wir wieder im Grundprogramm.

Was ist also zu tun, um wirklich Einzelschritte zu erreichen? Erinnern wir uns zurück an das Optionenmenue. Dort gab es einen sogenannten Debug-Mode, und genau den müssen wir jetzt aktivieren. Also nochmal zurück, und im Optionenmenue die Funktion „4=Zeichen/Zeile & Debug" aufrufen. Das System bietet uns dann die Möglichkeit „3=Debug-Info AN", genau die wollen wir aufrufen.

Anschließend ist es notwendig, unser Programm nochmal mit dem Assembler zu übersetzen, erst dann begeben wir uns wieder in unser Einzelschrittmenue. Wie vorhin rufen wir „3=Einzelschritt" auf und geben dann unser Symbol „START" ein. In der letzten Bildschirmzeile erkennen wir unseren per Editior eingegebenen Quelltext wieder (*Abb. 2.1.34*). Wenn man „CR" eingibt, so wird nur dieser eine Befehl ausgeführt; es ist also jetzt möglich, wirklich im Einzelschritt, gewissermaßen Schritt für Schritt die Funktion des Programms auszutesten.

Abschließend noch der Hinweis auf weitere Eingabemöglichkeiten. Während man im Einzelschritt ist, kann man verschiedene Kommandos geben. So kann man den Einzelschritt jederzeit durch die Eingabe von „M" verlassen, „L" steht für Bildschirmlöschen und „S" in Verbindung mit einer Ziffer zwischen 0 und 3 für den Wechsel der Bildseite. Diese Funktion wird später noch interessant für uns sein. Die Ziffer gibt dabei eine bestimmte Bildseite an, mit „S" und „CR" alleine erreichen wir den vor dem Aufruf vorhanden Zustand wieder. Falls ein Programmteil nicht in Einzelschritten, sondern in einem Durchlauf abgearbeitet werden soll, so stehen uns dafür weitere Möglichkeiten

```
Einzelschritt

Adr:

S=Seite N=n mal B=Bis L=Loeschen M=Menue
```

Abb. 2.1.31
Aufruf des Einzelschritt-Menues

```
D0       D1       D2       D3       D4       D5       D6       D7
FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFF7FD FFFFFFFF FFFFEFFF FF7FFFFF
A0       A1       A2       A3       A4       A5       A6       A7
FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF 00008000 FFF7FFFF 0000BFF0
SR   USP      SSP      PC
A700 0000BFF0 0000BFF0 00009C00 303C
```

Abb. 2.1.32
Bildschirminhalt bei Einzelschritt

```
D0       D1       D2       D3       D4       D5       D6       D7
00000000 FFFF0100 FFFF00B2 FFFFFFFF FFFFF7FD FFFFFFFF FFFFEFFF FF7FFFFF
A0       A1       A2       A3       A4       A5       A6       A7
FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF 00008000 FFF7FFFF 0000BFF0
SR   USP      SSP      PC
2704 0000BFF0 0000BFF0 00009C0A 4E75
```

Abb. 2.1.33
Ausführung eines Programms

```
D0       D1       D2       D3       D4       D5       D6       D7
00000000 FFFF0100 FFFF00B2 FFFFFFFF FFFFF7FD FFFFFFFF FFFFEFFF FF7FFFFF
A0       A1       A2       A3       A4       A5       A6       A7
FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF 00008000 FFF7FFFF 0000BFF0
SR   USP      SSP      PC
A700 0000BFF0 0000BFF0 00009C00  move #100,d0
```

Abb. 2.1.34
Einzelschritt mit „Debug Info an"; die Textzeile aus dem Editor wird mit angezeigt

zur Verfügung. So ist es mit dem Aufruf „N" in Verbindung mit einer Zahl möglich, n Schritte auszuführen. Mit „B" kann man eine Adresse eingeben, bei der die Ausführung dann gestoppt werden soll. All diese Möglichkeiten kann man aber erst nutzen, wenn mehr über die Programmierung bekannt ist. Das hier sollte nur eine knappe Darstellung der Funkionen sein, die später als Nachschlageteil verwendet werden kann.

2.2 Der Texteditor

Den Texteditor haben wir schon kurz kennengelernt. Er enthält jedoch eine Vielzahl von Befehlen, so daß ihm nur ein eigener Abschnitt gerecht werden kann.

Bevor wir uns jetzt mit den Befehlen des Editors auseinandersetzen, sei an dieser Stelle nochmal an die Bedienung der CTRL-Taste erinnert. Wir erinnern uns, daß zur Eingabe einer Kontrollfunktion die CTRL-Taste gleichzeitig mit einem Buchstaben zu drücken war. Um dies jetzt auf dem Bildschirm darstellen zu können, verwenden wir den Hochpfeil vor dem entsprechenden Buchstaben. Lesen wir auf dem Bildschirm z.B. ↑J, so bedeutet das, daß zum Auslösen der Funktion CTRL und „J" gleichzeitig zu drücken ist. Für Besitzer eines Grundprogramms ab der Version 4.3 ist es dabei gleichgültig, ob der eingegebene Buchstabe ein Groß- oder ein Kleinbuchstabe ist. Bei älteren Versionen muß ein Großbuchstabe eingegeben werden.

Jetzt wird es aber Zeit, daß wir den Editor aufrufen. Nachdem das bereits bekannte, leere Editorfenster sichtbar ist, geben wir CTRL-J ein. Die *Abb. 2.2.1* erscheint. Mit dieser Hilfefunktion kann man sich alle vorhandenen Befehle in Kurzform erklären lassen. Drückt man nun noch eine Taste, so erscheint *Abb. 2.2.2*, das ist gewissermaßen die zweite Seite der Hilfefunktion. Ein weiterer Tastendruck bringt uns wieder ins Editorfenster zurück.

Wie kann man sich die vielen Befehle merken? Sehen wir uns dazu die *Abb. 2.2.3* an. Dort ist ein Ausschnitt der Tastatur gezeigt. Die Steuerzeichen sind nun so gelegt, daß die Position auf der Tastatur mit der Bewegungsrichtung des Cursors auf dem Bildschirm übereinstimmt.

Zunächst die Cursorsteuerung. Mit den Befehlen:

CTRL-S, CTRL-D, CTRL-E und CTRL-X

kann man den Cursor auf dem Bildschirm frei bewegen.

Um diese Funktionen austesten zu können, tippen wir den Text in *Abb. 2.2.4* ein. Dabei achten wir wieder darauf, daß die Karozeichen links überschrieben werden, sonst wird der Text vom Editor nicht angenommen.

Nach der Eingabe steht der Cursor am Textende (Abb. 2.2.4). Versuchen wir also jetzt unsere bereits besprochenen Befehle für die Cursorsteuerung anzuwenden. Zunächst geben wir CTRL-E ein, und der Cursor wandert ein Zeichen nach oben (*Abb. 2.2.5*). Jetzt versuchen wir es nochmals mit der gleichen Funktion, also nocheinmal CTRL-E eingeben. Der Cursor kann eigentlich nicht weiter nach oben, daher wird das Bildfenster um eins nach unten geschoben. Es ergibt sich *Abb. 2.2.6*. Da sich oben kein Text mehr befindet, wandert der Cursor dabei zusätzlich nach links, rechts neben das Karofeld.

```
REV4.3 -1-
↑E=Zeile hoch      ↑X=Zeile runter
↑S=Zeichen links   ↑D=Zeichen rechts
↑A=Wort links      ↑F=Wort rechts
↑Z=Zeile auf       ↑W=Zeile ab
↑Y=Zeile loeschen  ↑N=Zeile einfuegen
↑T=Loeschen rechts
↑R=Seite zurueck   ↑C=Seite vor
↑V=Einfuegemode    ↑P=Zeichensatz
↑QC=an Textende    ↑QR=an Textanfang
↑QF=suchen         ↑QA=ersetzen
↑G,DEL=Zeichen loeschen
```

Abb. 2.2.1
Das Hilfs-Menue erscheint nach Drücken der Tasten CTRL und „J" im Editor

```
REV4.3 -2-
↑L=wiederhole ↑QF,↑QA

↑KX=ENDE

↑KB=Block Anf.  ↑KK=Block Ende
↑KY=Block loeschen
↑KV=verschieben ↑KC=kopieren
↑KR=lesen       ↑KW=speichern
↑KH=Marken loeschen
```

Abb. 2.2.2
Zweite Seite des Hilfs-Menues

Abb. 2.2.3 Die Belegung der Tasten, während der Benutzung des Editors

Abb. 2.2.4
Beispieltext zur Eingabe; der Cursor steht am Textende

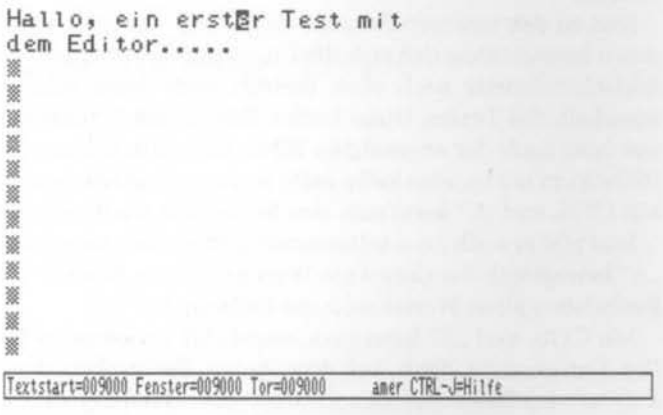


Abb. 2.2.5
Nach Eingabe von CTRL und „E"; der Cursor steht eine Zeile höher

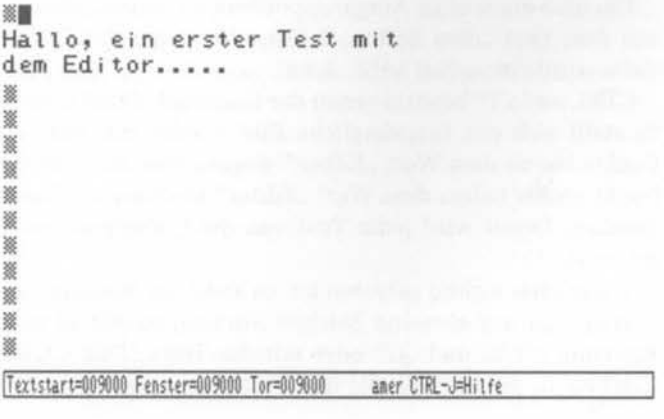


Abb. 2.2.6
Eine weitere Eingabe von CTRL und „E"; der Cursor wandert nicht höher, sondern der Bildschirminhalt wird nach unten geschoben

Nun CTRL und „X" drücken. Der Cursor bewegt sich um eine Zeile nach unten. Wenn man den Cursor ganz weit nach unten schieben würde, bis über den unteren Bildrand hinaus, so würde das Bildfenster wieder nach oben bewegt.

Jetzt mal CTRL und „S" drücken, der Cursor wandert eine Position nach links und blinkt dann über dem Buchstaben „H". Wenn man CTRL und „D" drückt, wandert er nach rechts. Hier gilt: Der Cursor steht immer nur über gültigem Text. Wenn man also den Cursor zu weit nach links rückt, so landet er eine Zeile höher, ganz rechts, neben dem ersten gültigen Zeichen dieser Zeile, und wenn man ihn zu weit nach rechts bewegt, so gelangt er in die darunterliegende Zeile. Das Bildfenster wird auch wieder automatisch verschoben, wenn man an die obere oder untere Bildschirmgrenze gerät.

Sehen wir uns nach den ersten Versuchen mit der Cursorbewegung nochmal die Abb. 2.2.3 an. Wie bereits gesagt, entspricht die Anordnung der Buchstaben der Cursorbewegung. Also der Buchstabe „E" liegt oben für eine Cursorbewegung nach oben, dementsprechend liegt das „X" unten, da mit CTRL und „X" der Cursor nach unten bewegt wird. „S" ist links und „D" rechts daneben. So kann man sich die Funktion leicht merken.

Nun zu den weiteren Befehlen. Mit CTRL und „W" wird das Bildschirmfenster nach unten bewegt, ohne daß sich die Cursorposition verändert. Mit CTRL und „Z" wird das Bildschirmfenster nach oben bewegt; auch dabei behält der Cursor seine Position innerhalb des Textes. Diese beiden Befehle sind nützlich, wenn man sehen will, was vor oder nach der angezeigten Bildschirmseite kommt. Mit CTRL und „R" wird der Bildschirm um ca. eine halbe Seite nach unten geschoben. *Abb. 2.2.7* zeigt die Wirkung. Mit CTRL und „C" kann man den Bildschirm um eine halbe Seite nach oben bewegen.

Jetzt gibt es noch zwei interessante Tasten, die ganze Wörter erkennen. Mit CTRL und „A" bewegt sich der Cursor ein Wort nach links. Er steht dann entweder auf dem ersten Buchstaben eines Wortes oder am Zeilenende.

Mit CTRL und „F" kann man umgekehrt um einzelne Wörter nach rechts schreiten. Der Cursor steht dann auf dem ersten Buchstaben des nächsten Wortes oder am Zeilenende. Das Programm erkennt die Trennung in einzelne Wörter anhand des Zwischenraums.

Bis hierher haben wir mit unseren Versuchen lediglich den Cursor oder das Bildschirmfenster verschoben. Jetzt wollen wir uns daran machen, den Text zu verändern.

Um eine eindeutige Ausgangsposition zu haben, bringen wir den Cursor in die Zeile mit dem Text „dem Editor...". Dann lösen wir die Funktion CTRL-N aus. Eine leere Zeile wurde eingefügt (*Abb. 2.2.8*).

CTRL und „Y" bewirkt genau das Gegenteil; damit läßt sich eine ganze Zeile löschen. Es stellt sich das ursprüngliche Bild wieder ein. Jetzt wollen wir die überflüssigen Punkte hinter dem Wort „Editor" weglöschen. Dazu wird der Cursor auf den zweiten Punkt rechts neben dem Wort „Editor" positioniert. Dann die Tasten CTRL und „T" drücken. Damit wird jeder Text von der Cursorposition bis zum rechten Zeilenende gelöscht.

Wenn alles richtig gelaufen ist, so sieht der Bildschirm jetzt wie die *Abb. 2.2.9* aus.

Will man nur einzelne Zeichen löschen, so gibt es zwei Möglichkeiten. Durch die Funktion CTRL und „G" oder mit der Taste „DEL". CTRL und „G" löscht man das Zeichen an der Cursorstelle und schiebt den restlichen Text der Zeile nach. Die Taste

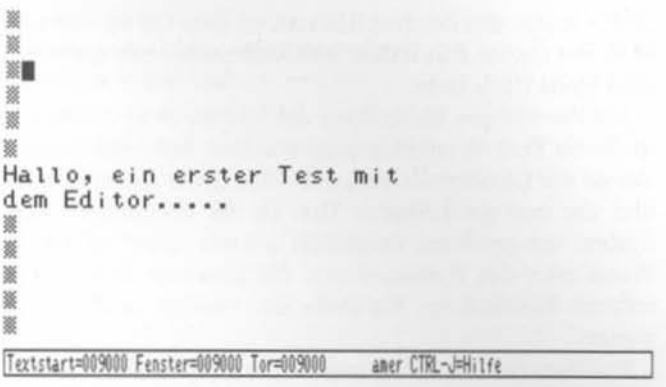


Abb. 2.2.7
Mit CTRL und „R" wird der Bildschirm um ca. ½ Seite nach unten geschoben

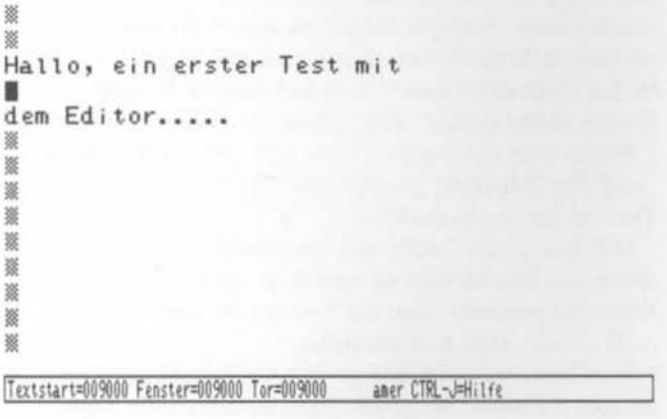


Abb. 2.2.8
CTRL und „N" fügen eine Zeile ein

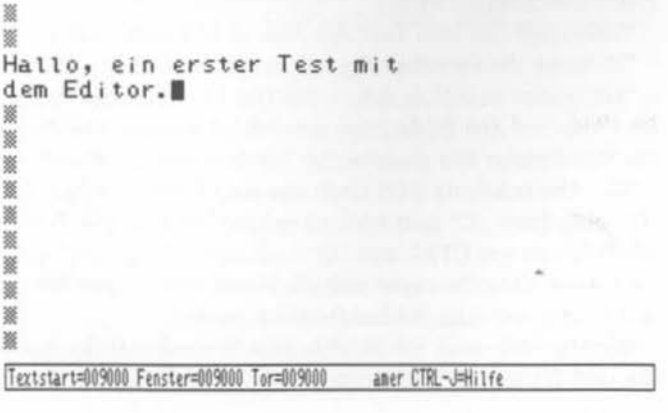


Abb. 2.2.9
CTRL und „Y" löschen eine Zeile

„DEL" löscht das Zeichen links neben dem Cursor und schiebt ebenfalls den Rest nach. Falls der Cursor den linken Bildrand noch nicht erreicht hat, bewegt er sich dabei um eine Stelle nach links.

Mit der jetzigen Einstellung des Editors ist es möglich, den Cursor an eine beliebige Stelle im Text zu positionieren und dort den bestehenden Text zu überschreiben. Soll der an der Cursorstelle stehende Text nicht überschrieben, sondern nur weitergerückt und der neu geschriebene Text an der Cursorstelle eingefügt werden, so kann das System entsprechend umgestellt werden. Dafür ist die Funktion CTRL-V vorgesehen. Wenn man das System durch die genannte Funktion umschaltet, so erscheint am unteren Bildrand zur Kontrolle die Anzeige „einf.". Dazu wollen wir einen Versuch starten.

Positionieren Sie den Cursor auf den Buchstaben „E" des Wortes „Editor". CTRL und „V" drücken, wenn es noch nicht eingeben wurde. Dann das Wort „Text-" eingeben. *Abb. 2.2.10* zeigt das Ergebnis. Der Text „Editor." wurde nach rechts verschoben.

Wie wir noch von der Besprechung des Optionenmenues wissen, ist es möglich, den Editor mit 40 oder mit 80 Zeichen pro Zeile zu betreiben. Texte, die durch das eben beschriebene Einfügen länger als eine Zeile werden, werden nicht automatisch in die nächste Zeile geschoben. Wenn man mit 40 Zeichen pro Zeile arbeitet, so wird der Text in den nicht sichtbaren Teil verschoben und bleibt erhalten. Erst wenn die 80 Zeichen-Grenze überschritten wird, gehen die Zeichen verloren.

Wenn man die Tasten CTRL und „V" erneut drückt, so verschwindet die Anzeige „einf" im Statusfeld wieder, und Zeichen, die man eintippt, überschreiben den alten Text an der Cursorposition.

Mit dem Editor lassen sich amerikanische und deutsche Zeichen eingeben. Um den deutschen Zeichensatz zu erreichen, muß man die Tasten CTRL und „P" eingeben. Zur Kontrolle wechselt dann die Anzeige im Statusfeld von „amer" auf „deut". Die Zeichen Ä, Ö , Ü etc. sind nun verfügbar.

Man kann deutsche Zeichen auch gemischt mit amerikanischen Zeichen, wie z.B. der eckigen Klammer, eingeben. Dazu muß man die Tasten CTRL und „P" erneut drücken, um wieder auf den amerikanischen Zeichensatz umzuschalten. Intern werden die deutschen Zeichen von den amerikanischen dadurch unterschieden, daß das Bit 7, also das höchste Bit innerhalb des Bytes, als Hinweis gesetzt wird. Dieses Bit verwendet der ASCII-Zeichensatz nicht.

Versuchen Sie, den Text der *Abb. 2.2.11* einzugeben.

Übrigens, die Berarbeitung von deutschen Zeichen macht den Editor langsamer, als er es mit reinen amerikanischen Zeichen ist, denn alle deutschen Zeichen müssen Punkt für Punkt auf den Bildschirm gezeichnet werden. Das dauert natürlich etwas länger als die Wiedergabe fest eingebauter Zeichen des Graphik-Prozessors.

Zur Abwechslung jetzt noch ein paar Cursorbefehle. Werden die Tasten CTRL und „Q" und dann „C" gedrückt, so gelangt man an das Textende. Umgekehrt wird durch das Drücken von CTRL und „Q" und dann „R" an den Textanfang positioniert. Hilfreich sind diese Einrichtungen natürlich erst, wenn man mit längeren Texten arbeitet, die nicht mehr auf eine Bildschirmseite passen.

Wichtig sind auch die Befehle zum Suchen und Ersetzen von Textteilen. Dazu setzen Sie den Cursor an den Anfang des Wortes „Hallo". Dann die Tasten CTRL und „Q"

```
Hallo, ein erster Test mit
dem Text-Editor.
```

```
Textstart=009000 Fenster=009000 Tor=009000 einf amer CTRL-J=Hilfe
```

Abb. 2.2.10
Nach CTRL und „V" erscheint der Text „einf" unten in der Status-Zeile, man kann dann Zeichen einfügen

```
Hallo, ein erster Test mit
dem Text-Editor.
Man kann deutsche Zeichen "Ä Ö Ü äöü ß"
mit amerikanischen Zeichen "[ \ ] {|} ¬"
gemischt eingeben.
```

```
Textstart=009000 Fenster=009000 Tor=009000 einf amer CTRL-J=Hilfe
```

Abb. 2.2.11
Mit CTRL und „P" kann man den Zeichensatz umschalten, dann ist die Eingabe von Umlauten möglich

```
Suche

Zeichen

Ersetze

Buchstaben
```

Abb. 2.2.12
Suche „Zeichen" und ersetze durch „Buchstaben"

gefolgt von „F" drücken. Es erscheint ein Eingabefeld. Dort gibt man den zu suchenden Text ein. Wir geben einmal die Buchstaben „ei" ein. Dann wird die Taste „CR" gedrückt. Der Cursor blinkt nun direkt hinter der gefundenen Buchstabengruppe, er befindet sich also in der ersten Zeile über dem „n" des Wortes „ein".

Betätigt man nun CTRL und „L", so wird die nächste Buchstabengruppe gesucht. Der Cursor bleibt also nach dem „ei" des Wortes „Zeichen" in der dritten Zeile stehen. Man kann nun von jeder Stelle aus die Suche mit CTRL und „L" fortfahren lassen. Der Suchtext bleibt solange gespeichert, bis man über CTRL-F ein neues Such- oder Ersetzkommando eingibt.

Es wird immer von der aktuellen Cursorstelle bis zum Textende gesucht. Wird die Buchstabenkombination nicht gefunden, so blinkt der Cursor auf dem ersten Karo nach dem Textende.

Nun bitte CTRL „Q" und „R" eingeben, wie wir es einige Zeilen weiter oben bereits kennengelernt haben. Der Cursor blinkt dann wieder auf dem ersten Buchstaben von „Hallo". Wir probieren jetzt den Befehl zum Austausch von Texten aus. Dazu wird CTRL und „Q" und dann „A" eingegeben.

Es erscheint ein Eingabefeld. Wir geben den Text „Zeichen" ein und bestätigen ihn mit „CR". Das ist der Text, nach dem gesucht werden soll. Jetzt geben wir noch den Text „Buchstaben" ein. Das ist der Text, durch den der gesuchte zu ersetzen ist. *Abb. 2.2.12* zeigt den Bildschirm. Nun drücken Sie die Taste „CR". Diesmal blinkt der Cursor beim ersten Buchstaben des gefundenen Textes, wie *Abb. 2.2.13* zeigt. Der Editor wartet nun auf die Bestätigung dafür, daß man den Text wirklich ersetzt haben will. Dazu muß man den Buchstaben „J" für „ja" eingeben. Das tun wir, und aus dem Wort „Zeichen" wurde „Buchstaben". Der Cursor blinkt bereits beim nächsten gefundenen Wort „Zeichen".

Wenn man einen Text nicht ersetzt haben will, so gibt man den Buchstaben „N" ein. Wird kein Text mehr gefunden, blinkt der Cursor am Schluß des gesamten Textes auf dem nachfolgenden Karo.

Will man die Suche abbrechen, so genügt es, irgendeine andere Taste als „N" oder „J" zu drücken.

Wie bereits bei der Suche durch CTRL-F, so läßt sich auch die CTRL-A-Funktion durch CTRL-L wiederholen.

Nun zu den komplizierteren Befehlen, wie sie in Abb. 2.2.2 zu sehen waren. Es handelt sich um die sogenannten Block-Befehle.

Mit der Tastenfolge CTRL-K und „B" wird der Anfang und durch die Eingabe von CTRL-K und „K" das Ende eines Textblocks markiert. Der Text, der zwischen diesen beiden Stellen liegt, wird für die anschließend beschriebenen Befehle als zusammenhängender Block betrachtet.

Zunächst markieren wir uns einen Block für unsere weiteren Versuche. Die *Abb. 2.2.14* zeigt den Bildschirm mit einem bereits markierten Block. Der nach rechts zeigende Pfeil steht dabei für den Anfang, der nach links zeigende für das Ende des Textblockes. Um diese Marken in Ihr System zu übertragen, bewegen Sie den Cursor an den gewünschten Anfang, geben CTRL-B ein, fahren weiter zum zu markierenden Ende und betätigen CTRL-K. Mit CTRL-K und „Y" wird der markierte Block gelöscht. Die *Abb. 2.2.15* zeigt den Bildschirm nach dieser Prozedur. Der Block darf sich natürlich auch über mehrere Bildseiten ausdehnen. Nach dem Löschen blinkt der Cursor am

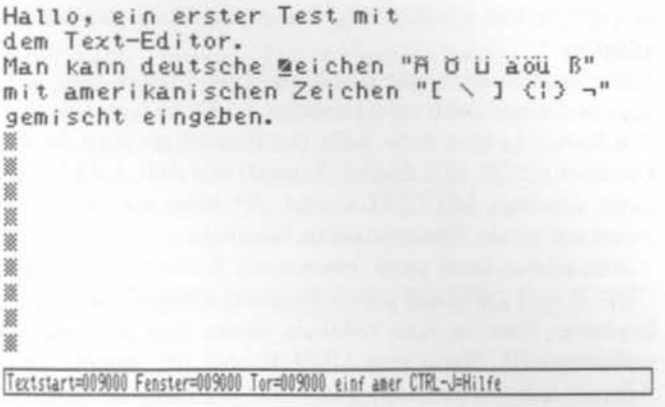


Abb. 2.2.13
„Zeichen" wurde gefunden

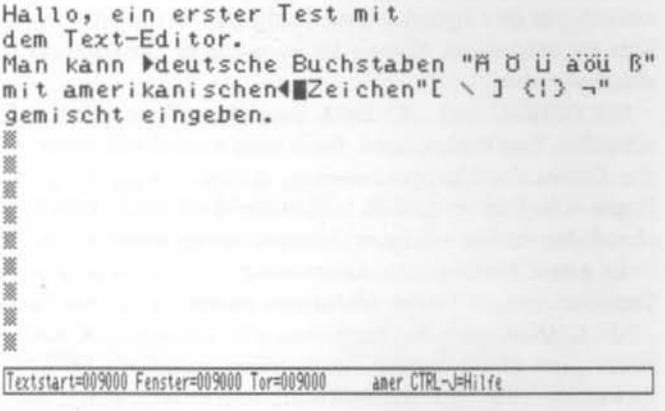


Abb. 2.2.14
Ein Block wurde markiert

Abb. 2.2.15
Der Block wurde gelöscht

ursprünglichen Blockanfang. Das ganze Bild kann man sich z.B. mit CTRL-R wieder ansehen.

Ein anderer wichtiger Blockbefehl ist CTRL-K und „C", mit dem man einen Block kopieren kann. Dazu wird zunächst der Block laut *Abb. 2.2.16* markiert. Wenn man nun den Cursor in eine neue Zeile positioniert, so wird der Block nach dem Auslösen der Funktion (CTRL-KC) dorthin kopiert, wie *Abb. 2.2.17* zeigt. Die Markierungen bleiben dabei erhalten. Mit CTRL-K und „V" kann man einen Block transportieren. Er wird dabei zur neuen Cursorposition verschoben.

Nun gibt es noch zwei interessante Befehle. Block-Lesen und Block-Speichern. Mit CTRL-K und „W" kann man einen markierten Block in ein anderes RAM-Speichergebiet kopieren. Dies ist sehr nützlich, wenn man ein sehr langes Programm in kleinere aufteilen will. Wenn man CTRL-K und „W" drückt, so wird man zur Eingabe einer Adresse aufgefordert.Man kann eine beliebige Adresse innerhalb des RAM-Speichers angeben. Die bereits besprochenen Kriterien für die Auswahl von freiem Speicher sind dabei natürlich zu beachten.

Mit dem Optionenmenue kann man dann z. B. den Textstart auf den neuen Bereich stellen, um den ausgelagerten Textteil nach einem erneuten Aufruf des Editors gesondert zu bearbeiten. Ebenso ist es möglich, diesen Text durch CAS-Schreiben direkt abzuspeichern.

Mit CTRL-K und „R" kann man einen Text aus dem Speicher lesen und in den aktuellen Text einkopieren. Dazu muß man die Blockmarken nicht setzen, sondern nur den Cursor dorthin positionieren, wo nach dem Aufruf der Funktion der Blockanfang liegen soll. Das Programm fragt dann nach der Quell-Adresse, die man dann entsprechend der vorher erfolgten Abspeicherung angeben muß. Das Programm erkennt das Ende eines Textbereichs automatisch. Zur Kennzeichnung wird der Code 00 an das Textende gesetzt. Dieses Markieren erfolgt durch den Editor automatisch.

Mit CTRL-K und „H" kann man alle Blockmarken entfernen. Das braucht man z. B., wenn man einen neuen Textbereich markieren will, aber noch alte Blockmarken vorhanden sind. Um falsche Reaktionen zu vermeiden, empfiehlt es sich, die Blockmarken immer nach Gebrauch zu entfernen. Beim Verlassen des Editors wird das auch automatisch getan.

Abschließend noch kurz etwas zur Arbeitsweise des Editors.

Abb. 2.2.18 zeigt eine schematische Darstellung des Speichers. Der Textanfang ist normalerweise nach dem Einschalten auf die Adresse $9000 gestellt. In der unteren Zeile des Editorfensters sind dazu Angaben zu finden. Der Textstart steht dort hinter der Angabe „Textstart=".

Das Bildschirmfenster muß aber nicht den Inhalt des unmittelbar folgenden Textes zeigen, sondern es beginnt bei der Adresse „Fenster=", die ebenfalls unten am Bildschirmrand erscheint.

Die Angabe „Tor=" gibt die letzte aktuell belegte Speicherzelle an. Zu beachten ist dabei, daß der gerade im Fensterbereich stehende, also sichtbare Text nicht mit zu diesem Bereich zählt, sondern in einem gesonderten Speicher steht, der vom Grundprogramm als Bildwiederholspeicher verwendet wird. Wenn man die Endadresse des gesamten Textes erfahren will, so muß man z.B. mit CTRL-Q und „C" oder mit CTRL-C das Bildfenster solange verschieben, bis kein Text mehr auf dem Bildschirm sichtbar ist,

```
Hallo, ein erster Test mit
dem Text-Editor.
▶Man kann  Zeichen"[ \ ] {¦} ¬"
gemischt eingeben.◀
```

Textstart=009000 Fenster=009000 Tor=009000 amer CTRL-J=Hilfe

Abb. 2.2.16
Markieren eines Blockes

```
Hallo, ein erster Test mit
dem Text-Editor.
Man kann  Zeichen"[ \ ] {¦} ¬"
gemischt eingeben.

▶Man kann  Zeichen"[ \ ] {¦} ¬"
gemischt eingeben.◀
```

Textstart=009000 Fenster=009000 Tor=009000 amer CTRL-J=Hilfe

Abb. 2.2.17
Der Block wurde kopiert

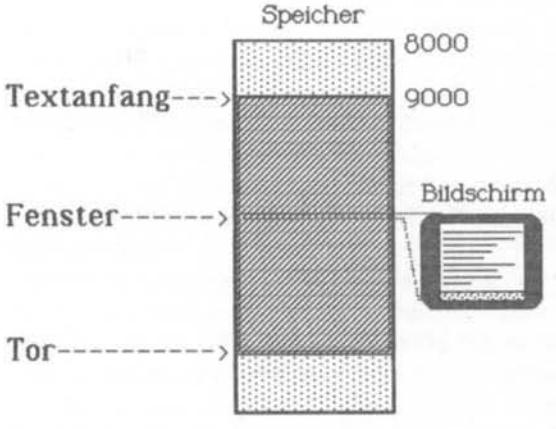


Abb. 2.2.18 Aufteilung des Speichers beim Editor

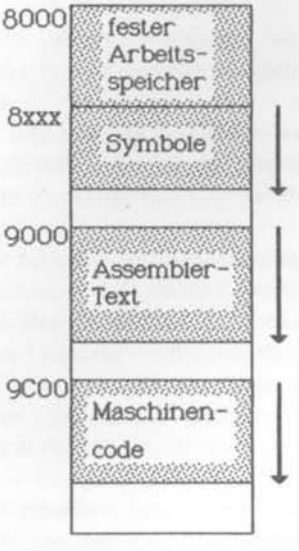


Abb. 2.2.19 Gesamte Speicheraufteilung beim Grundprogramm

sondern nur noch die Karo-Zeichen. Dann gibt „Tor=" die letzte Adresse an, die der gesamte Text belegt. Dort steht dann auch der Code 00, der das Textende markiert.

Warum ist diese Angabe so wichtig? Sehen wir uns dazu die Speicheraufteilung in Abb. 2.2.19 an. Das Grundprogramm verwaltet mehrere Speicherbereiche. Bei Adresse $8000 beginnt der Arbeitsspeicher des Grundprogramms. Dahinter liegt die Symboltabelle, mit den durch den Benutzer frei definierten Namen. Dann kommt eine Lücke. Der Text beginnt normalerweise, also falls keine Veränderung durch den Aufruf des Optionenmenues vorgenommen wurde, bei Adresse $9000. Das Textende liegt bei „Tor=" und ist somit flexibel, also von der Größe des eingegebenen Textes abhängig.

Ab Adresse $9C00 wird standardmäßig, also falls keine weitere Zusatzanweisung (durch ORG ...) erfolgte, das übersetzte Maschinenprogramm abgelegt. Wir sehen schon, was da passieren kann. Wenn mehr Text eingegeben wird, als bis zum Beginn des Maschinencodes Platz findet, die Adresse „Tor=" also größer ist als der Anfang des Maschinenprogramms, so wird während des Assemblierens ein Teil des Textes überschrieben. Daher sollte man bei langen Programmen auf die Angabe „Tor=" achten. Ebenfalls kann die Symboltabelle in den Textbereich laufen. Dazu wird bei einer Übersetzung die Endadresse der Symboltabelle vom Assembler ausgegeben. Wenn man Programme mit vielen Symbolen hat, sollte man sicherheitshalber den Textanfang hoch ansetzen. Das gleiche gilt für den Start des Maschinenprogramms.

3 Versuche mit dem Grundprogramm

Anhand zahlreicher Versuche werden in diesem Kapitel Befehle des 68000/8 vermittelt. Ein wichtiger Aspekt ist aber auch die systematische Fehlersuche und die Umsetzung von Problemen in Programme.

3.1 Die Zeichensprache

Die Zeichensprache stellt eigentlich keine eigene Programmiersprache, sondern eine Kombination aus der Programmierung in Assembler und der Verwendung, also dem Aufruf von bereits vorliegenden Unterprogrammen dar. Die Unterprogramme tragen dabei Namen, die eindeutig die Funktion erkennen lassen. Wie der Name „Zeichensprache" bereits erkennen läßt, sind diese Unterprogramme in erster Linie für Bildschirmarbeit entstanden. Es handelt sich dabei gewissermaßen um eine graphische Sprache. Durch die Verwendung dieser fertigen Programmteile ist es möglich, in Assembler programmieren zu lernen und dabei von Anfang an ansprechende Resultate auf dem Bildschirm zu sehen.

Die Prozessoren 68000 und 68008 besitzen die in der *Abb. 3.1.1* dargestellten Register. Es gibt acht Datenregister und acht sogenannte Adreßregister. Diese Register sind 32 Bit breit, können also Werte im Bereich von 0 bis $ffffffff aufnehmen.

Die Datenregister können auch als 8- oder 16-Bit-Register verwendet werden. Arithmetische Operationen wie Addition, Multiplikation etc. werden in den Datenregistern ausgeführt. Alle Datenregister sind dabei gleichberechtigt.

Die Adreßregister haben dagegen eine Sonderfunktion. Wie man sie anwendet, wird im Verlauf des Kapitels noch gezeigt. Innerhalb der Adreßregister nimmt das Register A7 nochmals eine Sonderstellung ein. Es übernimmt die Funktion des sogenannten Stackpointers.

Durch den Programmzähler wird angezeigt, an welcher Adresse der Maschinencode gerade ausgeführt wird. Im Statusregister finden sich Hinweise auf Überlauf, Vorzeichen etc., wie sie bei arithmetischen Operationen und Vergleichen gebraucht werden.

Doch fangen wir einmal ganz einfach an. Beginnen wir mit unserer bereits besprochenen Zeichensprache. Dazu stellen wir uns eine Schildkröte vor, die wir über den Bildschirm bewegen können. Diese Schildkröte ist mit einem Schreibstift versehen und zeichnet darum bei ihrer Wanderung eine Linie. Sie wird auf dem Bildschirm durch ein Dreieck symbolisiert.

Mit einem Befehl kann man die Schildkröte vorwärts schreiten lassen, mit einem anderen wird sie gedreht. Diese Befehle sind Beispiele für die bereits erwähnten Unterprogramme im Grundprogramm.

Die Schildkröte zeichnet

Dazu einen einfachen Versuch. *Abb. 3.1.2* zeigt das Programm. Tippen Sie das Programm mit Hilfe des Editors ein, so wie Sie es im Kapitel 2 gelernt haben.

Der Editor wird dann mit CTRL-K und „X" verlassen. Man kommt dann wieder in das Grundmenue, das den Aufruf des Assemblers beinhaltet. Es wird nun „2=Assembler" aufgerufen. Als Ergebnis der Übersetzung sollte nun die *Abb. 3.1.3* auf dem Bildschirm zu sehen sein. Der Assembler hat das eingegebene Programm also in den Maschinencode übersetzt. In der rechten Spalte sieht man noch den eingegebenen Text, links daneben befindet sich die Übersetzung. Ganz links erscheinen die Adressen, bei denen der jeweilige Maschinencode im Speicher abgelegt wurde.

Die Taste „M" bringt uns zurück in das Grundmenue.

Dort wird der Befehl „2=starten" eingegeben. Dann fragt das Grundprogramm nach der Startadresse. Man gibt hier den Namen „START" ein. Durch das Bestätigen dieser Eingabe per „CR" wird das Programm gestartet. Der Name „START" wurde in unserem Programm in der ersten Zeile durch „START:" definiert.

Nach dem Ablauf des Programms erscheint die *Abb. 3.1.4*. Dabei sieht man auf dem Bildschirm aber zusätzlich die Schildkröte, also das Dreieck, mit eingeblendet. Es sitzt am Ende der gezeichneten Strecke. Wenn man das Dreieck nicht sehen will, denn das Bild flimmert dabei leicht, so drückt man die Taste „F" (F=Flip), und die Schildkröte verschwindet. Die Schildkröte steht nach dem Start immer in der Mitte des Bildschirms. Dabei zeigt sie nach oben.

Wie funktioniert unser Programm?

In der ersten Zeile steht, wie schon erwähnt, eine Namensdefinition. Der Doppelpunkt ist dabei wesentlich. Daran erkennt der Assembler, daß dem Namen, der davor steht, ein Wert zugewiesen werden soll. Dieser Wert ist die Adresse, bei der die Marke gerade steht. Sehen wir uns dazu die Assemblerausgabe an. In Abb. 3.1.3 steht ganz links der Wert 9C00 neben dem Namen „START". Das ist also der Wert von „START". Der Maschinencode wird beginnend bei dieser Adresse abgelegt. Nach dem Einschalten des Systems ist dieser Wert $9C00 als Standard eingestellt. Wenn man eine andere Adresse angeben will, so muß man das dem Assembler mitteilen, dafür gibt es die ORG-Anweisung, die wir später noch kennenlernen werden.

In der nächsten Zeile steht der Befehl „MOVE #100,D0".

Das ist ein 68000/8 Assemblerbefehl. Er bedeutet: Nimm den dezimalen Wert 100 und lege ihn in das Register D0 ab. Dieser Befehl belegt vier Bytes im Speicher, weshalb die nächste freie Adresse $9C04 ist, die für den nächsten Maschinencode verwendet wird. Dort steht der Befehl „JSR @SCHREITE". JSR bedeutet Jump Subroutine, oder zu deutsch, springe in ein Unterprogramm. Dieses Unterprogramm trägt den Namen „SCHREITE" und befindet sich in den EPROMs des Grundprogramms. Das Zeichen „@" wird vom Assembler benötigt, um Namen des Grundprogramms von eigenen

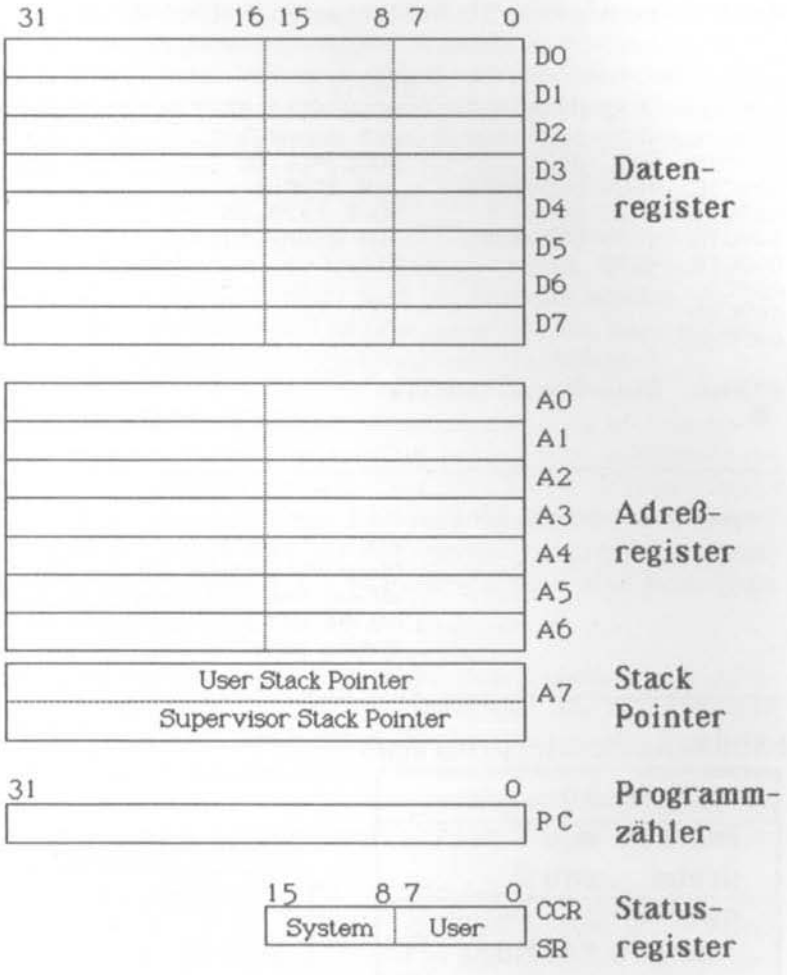


Abb. 3.1.1
Die Register
des 68000
und des 68008

```
start:
 move #100,d0
 jsr @schreite
 move #60,d0
 jsr @drehe
 move #120,d0
 jsr @schreite
 rts
```

Textstart=009000 Fenster=009000 Tor=009000 einf amer CTRL-J=Hilfe

Abb. 3.1.2
Das erste
Schildkrötenprogramm

```
Rolf-D.Klein 68000/08 Assembler 4.3 (C) 19E

009C00                          START:
009C00   303C 0064               MOVE #100,D0
009C04   4EB9 00001288           JSR @SCHREITE
009C0A   303C 003C               MOVE #60,D0
009C0E   4EB9 000012F6           JSR @DREHE
009C14   303C 0078               MOVE #120,D0
009C18   4EB9 00001288           JSR @SCHREITE
009C1E   4E75                    RTS
009C20
009C20
009C20

008A60   Ende-Symboltabelle
 ■                                        F=Flip M=Menue
```

Abb. 3.1.3
Der Assembler übersetzt es in Maschinensprache

Abb. 3.1.4 (rechts)
Dieses Bild erscheint nach dem Start auf dem Bildschirm, dabei ist zusätzlich die Schildkröte eingeblendet, die in dieser Darstellung fehlt

F=Flip M=Menue

68008-Assemblerprogramm

```
   org $9c00
schreite equ 1
drehe    equ 2
quadrat:
  move.w #4-1,d3
schleife:
  move.w #50,d0
  move.w #schreite,d7
  trap #1
  move.w #90,d0
  move.w #drehe,d7
  trap #1
  dbra d3,schleife
  rts

  end
```

Z80-Programm

```
QUADRAT:=$
 21 4.W
 CD SCHLEIFE
 21 #50.W
 CD SCHREITE

 21 #90.W
 CD DREHE

 CD ENDSCHLEIFE
 C9
```

Abb. 3.1.5 Die Befehle des 68000/68008 und des Z80 im Vergleich

Namen zu unterscheiden. Dieses Unterprogramm SCHREITE bewegt die Schildkröte um die durch den Inhalt des Registers D0 vorgegebene Anzahl an Schritten vorwärts.

Wenn man die Schildkröte drehen will, so verwendet man einen ähnlichen Mechanismus. Dazu wird hier der Wert 60 in das Register D0 geladen und anschließend das Unterprogramm mit dem Namen „DREHE" aufgerufen. Es dreht die Schildkröte um so viele Grade entgegen dem Uhrzeiger, wie im Register D0 angegeben sind.

Danach folgt nochmals ein Schreite-Befehl, so daß sich die *Abb. 3.1.4* ergibt. Der Befehl, der am Schluß des Programms steht, heißt „RTS" und bedeutet Return from Subroutine, oder zu deutsch: kehre aus dem Unterprogramm zurück. Denn hier endet unser Programmstück. Durch den RTS-Befehl weiß der Prozessor 68000/8, wo das Programm zu Ende ist, und setzt seine Arbeit im Grundprogramm fort. Unser gesamtes Programm wurde also vom Grundprogramm als Unterprogramm betrachtet.

Wir zeichnen ein Quadrat

Betrachten wir ein zweites Beispiel. Die *Abb. 3.1.5* zeigt das 68000/8 Assemblerprogramm und zum Vergleich ein entsprechendes Z80-Programm, wie es mit dem Grundprogramm der Z80-Version des NDR-KLEIN-Computers erstellt werden kann (siehe Mikrocomputer selbstgebaut und programmiert).

Die Art der Aufrufe wurde hier nochmals variiert.

In der ersten Zeile steht der ORG-Befehl. ORG ist von dem englischen Wort „originate" abgeleitet und bedeutet soviel wie „Ursprung". Damit kann man dem Assembler eine Startadresse für den Maschinencode vorgeben. Wenn man keine ORG-Anweisung angibt, ist die Adresse auf $9C00 festgelegt. Diesen Standardwert hatten wir in unserem letzten Beispiel verwendet. Dennoch schadet es nicht, wie hier, eine Anweisung „ORG $9C00" zu geben. Übrigens, Groß- und Kleinschreibung unterscheidet der Assembler nicht, man kann sie beliebig verwenden.

Die nächsten beiden Zeilen sind Namensdefinitionen. Dem Namen „SCHREITE" wird der Wert 1 zugeordnet und dem Namen „DREHE" der Wert 2. Dabei ist der Befehl „EQU", genauso wie die ORG-Anweisung ein Befehl an den Assembler. EQU kommt aus dem Englischen „equal", und damit ist eine Gleichsetzung gemeint. Immer wenn ab dieser Definition in unserem Programm eines der beiden definierten Wörter auftaucht, wird es durch den Assembler bei der Übersetzung durch die dafür festgelegte Ziffer ausgetauscht.

Jetzt folgt das eigentliche Programm.

Mit „QUADRAT:" wird dem Programmanfang der Name „QUADRAT" gegeben. Es erfolgt eine Wertzuweisung der Startadresse an den Namen „QUADRAT"; dies geschieht hier in gleicher Weise, wie im vorigen Beispiel mit dem Namen „START". Im Z80-Programm wurde dies durch „QUADRAT:=$" getan.

Um ein Quadrat zu zeichnen, wiederholen wir einfach viermal die Anweisungen SCHREITE VORWAERTS, DREHE um 90°. Um die Wiederholung vorzubereiten, wird

mit „MOVE.W #4-1,D3" der Wert 3 (!) in das Register D3 transportiert. Im Z80 Programm war dies der Befehl „21 4.W", der allerdings den Wert 4 in das Registerpaar HL transportiert hatte. Warum der Wert 3, wenn man viermal etwas tun will? Dazu sehen wir uns das Ende des Programms an. Vor dem RTS-Befehl steht der neue Befehl „DBRA D3,SCHLEIFE". Dieser Befehl bedeutet „decrement and branch". Also verringere und springe. Dabei ist D3 das verwendete Register, und hinter dem Komma steht ein Sprungziel. Das Register D3 wird von diesem Befehl um eins verringert. Ein Sprung zum angegebenen Ziel erfolgt immer dann, wenn das Register den Wert -1 noch nicht erreicht hat. Das Sprungziel ist in unserem Fall weiter oben durch „SCHLEIFE:" definiert.

Wenn man also das Register D3 mit dem Wert 3 lädt, so wird alles zwischen der Marke „SCHLEIFE" und dem „DBRA"-Befehl viermal durchlaufen. Denken wir diesen Vorgang einmal im einzelnen durch. Beim ersten Mal steht zunächst der Wert 3 im Register D3. Dann wird der Programmabschnitt dazwischen einmal durchlaufen, bis der „DBRA"-Befehl ausgeführt wird. Dieser verringert den Inhalt von D3 um eins, und es steht der Wert 2 im Register D3. Nun ist der Wert noch nicht -1, folglich wird ein Sprung nach „SCHLEIFE" ausgeführt. Das Programm wird jetzt zum zweiten Mal durchlaufen, kommt wieder bei „DBRA" an und verringert D3 erneut um eins. Jetzt steht der Wert 1 im Register. Wieder springt das Programm zur Marke „SCHLEIFE". Der Programmteil danach wird nun zum dritten Mal ausgeführt, und wieder wird D3 um eins verringert. Nun steht der Wert 0 in D3. Auch jetzt springt das Programm zurück zur Marke „SCHLEIFE", der Programmabschnitt wird zum vierten Mal ausgeführt. Diesmal wird D3 auf -1 verringert, und damit ist die Abbruchbedingung erfüllt, „DBRA" löst keinen Sprung nach „START" mehr aus. Der Prozessor setzt somit seine Arbeit beim nächsten Befehl, dem „RTS" fort. Wie uns bereits bekannt ist, wird damit unser Programm beendet. Nun zu den Befehlen dazwischen.

Mit dem „MOVE"-Befehl wird D0 mit dem Wert 50 geladen. Das ist der Wert für die Anzahl der auszuführenden Schritte. Nun könnte man das Unterprogramm „SCHREITE" auch mit dem „JSR„-Befehl aufrufen, den wir vorher schon mal kennengelernt haben, doch es geht auch noch anders. Dazu wird der Wert von „SCHREITE", der ja ganz oben mit der „EQU"-Anweisung definiert wurde, in das Register D7 geladen. Danach folgt ein neuer Befehl: „TRAP #1". Dieser TRAP-Befehl ruft ebenfalls das Grundprogramm auf. Im Register D7 wird dafür die Nummer des Unterprogramms erwartet.

Folgender Unterschied besteht zwischen TRAP und JSR: Beim JSR-Befehl wird die absolute Adresse des Unterprogramms eingesetzt. Bei uns z. B. $1288. Wenn Sie einmal einen Blick in die Abb. 3.1.3 werfen, so werden Sie erkennen, daß der Assembler diesen Wert in das Maschinenprogramm eingesetzt hat. Der TRAP-Befehl dagegen benötigt keine absolute Adresse. Er springt an eine feste Stelle im Grundprogramm. Dort wird dann das Register D7 zur Auswahl des Unterprogramms herangezogen. Hat es den Wert 1, so wird das Unterprogramm SCHREITE ausgeführt, bei 2 das Unterprogramm DREHE usw. Der Vorteil dieses Befehls liegt darin, daß ein Anwenderprogramm auch dann noch läuft, wenn sich die Unterprogrammadressen durch den Einsatz einer Grundprogrammversion verschieben. Wenn man eigene Programme entwickelt, die man weitergeben möchte, sollte man also dem TRAP-Befehl den Vorzug geben. Wird mit dem

Programm auch der Assemblercode weitergegeben, so ist es natürlich für den neuen Besitzer möglich, das Programm unter Verwendung der neuen Adressen neu zu übersetzen.

Eine weitere Besonderheit in unserem Programm ist das ".W", das Sie vielleicht schon skeptisch begutachtet haben. Es ist eine von drei Möglichkeiten zur Angabe, wieviel Bit man innerhalb des Registers belegen möchte. Die beiden anderen Möglichkeiten sind ".B" für ein Byte (8 Bit) und ".L" für ein Langwort von 32 Bit. Das Ganze hatten wir uns schon einmal im Kapitel 2 angesehen.

Läßt man die Angabe „." weg, so wird automatisch „.W" angenommen, also ein 16 Bit Datum verwendet. Wir sollten jedoch immer die Angabe „.W" eintippen, um bei einer eventuellen Fehlersuche eine Erleichterung zu haben.
Wir geben das Programm einmal ein. Die *Abb. 3.1.6* zeigt es im Editorfenster, und in der *Abb. 3.1.7* ist es nach der Übersetzung durch den Assembler zu sehen. Wenn man nur mit 40 Zeichen pro Zeile arbeitet, werden lange Zeilen einfach gekürzt. Will man das Assemblerlisting in voller Breite begutachten, so muß man mit Hilfe des Optionenmenues (siehe Kapitel 2) auf 80 Zeichen pro Zeile umschalten.

Wir starten nun das Programm. Dazu wird im Grundprogramm „2=starten" aufgerufen und bei der Startadresse der Name „QUADRAT" eingegeben. Wenn sich keine Fehler eingeschlichen haben, so sollte jetzt auf dem Bildschirm ein Quadrat erscheinen.

Noch ein paar Bemerkungen zu den Assemblerbefehlen. Beim „MOVE"-Befehl haben wir das Zeichen „#" vor den Zahlenwert gesetzt. Das Zeichen bedeutet: Verwende den nachfolgenden Wert direkt. Wenn man es wegläßt, so bedeutet das: Verwende nachfolgenden Wert als Adresse. In diesem Fall würde also nicht der angegbene Wert in das betreffende Register geladen, sondern der Wert, der sich an der angegebenen Adresse befindet. Die *Abb. 3.1.8* zeigt eine allgemeine Darstellung des MOVE-Befehls, wie wir ihn schon verwendet hatten. Achtung! Alle, die schon mal mit dem Z80 gearbeitet haben, werden feststellen, daß beim 68000/8 die Angabe von Ziel und Quelle genau umgekehrt sind. Hier steht die Quelle gleich hinter dem Befehl, und nach dem Komma folgt das Ziel. Beim Z80 Befehl „LD", der dem etwa entspricht, ist es umgekehrt.

Damit man nicht alle Namen von Unterprogrammen des Grundprogramms mit „EQU"-Anweisungen definieren muß, gibt es einen einfacheren Weg. Wenn man ein Ausrufezeichen vor das Symbol stellt, so wird es vom Assembler als Name eines Unterprogramms im Grundprogramm erkannt. Diesmal wird aber nicht die absolute Adresse eingesetzt, sondern der Index, also die Zahl, die man für den Trap-Aufruf benötigt. Die *Abb. 3.1.9* zeigt ein Beispiel.

Der Vollständigkeit halber ist in der *Abb. 3.1.10* noch die Variante mit absoluten Adressen, also mit dem Aufruf per JSR-Befehl zu sehen. Dort wird das Zeichen „@" als Erkennung verwendet. Die *Abb. 3.1.11* gibt das Bild nach dem Assembliervorgang wieder.

Leider klappt es beim Programmieren, bzw. beim Inbetriebnehmen eines Programmes nicht immer so reibungslos wie hier bei einem bereits ausgetesteten Programm. Ein wichtiger Punkt ist darum die Fehlersuche. Eine wesentliche Erleichterung stellt dabei ein gut kommentiertes Listing dar. Darum sollte man seine Programme stets kommentieren. Kommentare kann man direkt in das Assemblerprogramm hineinschreiben. Sie sollten Erklärungen und Hinweise zur Funktion des Programmes enthalten. Dabei

```
 org $9c00
schreite equ 1
drehe    equ 2
quadrat:
 move #4-1,d3
schleife:
 move.w #50,d0
 move.w #schreite,d7
 trap #1
 move.w #90,d0
 move.w #drehe,d7
 trap #1
 dbra d3,schleife
 rts

 end
```

Abb. 3.1.6
Das Quadratprogramm im Editorfenster

```
Textstart=009000 Fenster=009000 Tor=009000      amer CTRL-J=Hilfe
```

```
Rolf-D.Klein 68000/08 Assembler 4.3 (C) 19E


009C00                          ORG $9C00
= 00000001                      SCHREITE EQU 1
= 00000002                      DREHE    EQU 2
009C00                          QUADRAT:
009C00  363C 0003               MOVE #4-1,D3
009C04                          SCHLEIFE:
009C04  303C 0032               MOVE.W #50,D0
009C08  3E3C 0001               MOVE.W #SCHREITE D7
009C0C  4E41                    TRAP #1
009C0E  303C 005A               MOVE.W #90,D0
009C12  3E3C 0002               MOVE.W #DREHE,D7
009C16  4E41                    TRAP #1
009C18  51CB FFEA               DBRA D3,SCHLEIFE
009C1C  4E75                    RTS
009C1E
009C1E                          END

008AA8  Ende-Symboltabelle
```

Abb. 3.1.7 So übersetzt es der Assembler

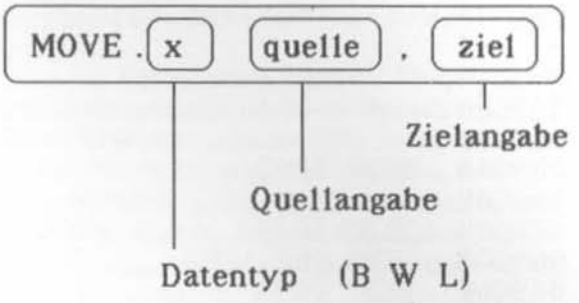


Abb. 3.1.8
Die allgemeine Form des „MOVE"-Befehls

```
 org $9c00
quadrat:
 move #4-1,d3
schleife:
 move.w #50,d0
 move.w #!schreite,d7
 trap #1
 move.w #90,d0
 move.w #!drehe,d7
 trap #1
 dbra d3,schleife
 rts

 end
```

Abb. 3.1.9 Wenn man das Zeichen „!" verwendet, braucht man die Symbole „schreite" und „drehe" nicht mehr zu definieren

```
Textstart=009000 Fenster=009000 Tor=009000 einf amer CTRL-J=Hilfe
```

```
 org $9c00
quadrat:
 move #4-1,d3
schleife:
 move.w #50,d0
 jsr @schreite
 move.w #90,d0
 jsr @drehe
 dbra d3,schleife
 rts

 end
```

Abb. 3.1.10 Mit „@" kann man die eingebauten Symbole auch als Adreßkonstante verwenden

```
Textstart=009000 Fenster=009000 Tor=009000 einf amer CTRL-J=Hilfe
```

```
Rolf-D.Klein 68000/08 Assembler 4.3 (C) 19E


009C00                          ORG $9C00
009C00                        QUADRAT:
009C00  363C 0003              MOVE #4-1,D3
009C04                        SCHLEIFE:
009C04  303C 0032              MOVE.W #50,D0
009C08  4EB9 00001288          JSR @SCHREITE
009C0E  303C 005A              MOVE.W #90,D0
009C12  4EB9 000012F6          JSR @DREHE
009C18  51CB FFEA              DBRA D3,SCHLEIFE
009C1C  4E75                   RTS
009C1E
009C1E                         END

008AA8  Ende-Symboltabelle
```

Abb. 3.1.11 Der Assembler erzeugt dann absolute Adressen im Maschinencode

```
                                                    F=Flip M=Menue
org $9c00
quadrat:
move #4-1,d3          # vier Mal
schleife:             # Schleifenanfang
move.w #50,d0         # Seitenlaenge
jsr @schreite         # vorwaerts
move.w #90,d0         # Winkelangabe
jsr @drehe            # und ausfuehren
dbra d3,schleife      # Schleifenende
rts                   # Programmende

end

Textstart=009000 Fenster=009000 Tor=009000 einf amer CTRL-J=Hilfe
```

Abb. 3.1.12 Das Quadrat-Programm mit Kommentaren

werden die Programme durch das Zeichen „*" oder auch wahlweise durch das Zeichen ";" vom Rest der Assemblerzeile abgetrennt.

Die *Abb. 3.1.12* zeigt unser Programm mit Kommentaren versehen. Hier wurde der Übersicht halber auf 80 Zeichen pro Zeile umgeschaltet (Optionen-Menue).

Zur Übung für den Ernstfall wollen wir die Fehlersuche mit Hilfe des Einzelschritt-Menues durchführen. Dazu schaltet man im Optionen-Menue „3=Debug-Info AN". Anschließend müssen wir den Assembler aufrufen. Im Gegensatz zu Assembliervorgängen ohne „Debug-Info" ist nun ganz unten im Bild die Angabe „Ende-Debug-Tabelle" und links daneben die Angabe der letzten verwendeten Adresse zu sehen. Diese Debug-Tabelle wird für das Einzelschritt-Menue gebraucht.

Damit sind die Vorbereitungen für unseren Einzelschrittbetrieb getroffen, und wir können das Menue „3=Einzelschritt" aufrufen. Dort wird nach einer Adresse gefragt. Wie vorhin beim normalen Programmstart geben wir den Namen „QUADRAT" ein. Mit „CR" gelangt man in den Einzelschritt. Wie in der *Abb. 3.1.13* ist auf dem Bildschirm jetzt die erste Zeile unseres Programms zu sehen. Auch die eingegebene Kommentarzeile wird zur besseren Orientierung mit angezeigt. Außerdem sind unten alle Registerinhalte des Prozesssors mit ausgegeben. Die Werte sind anfangs noch undefiniert, doch das wird sich gleich ändern.

Bis hierher haben wir nur unser Programm bearbeitet, also Vorbereitungen für den Einzelschrittbetrieb getroffen. Es wurde also noch kein einziger Befehl ausgeführt.

Wenn wir jetzt gleich die „CR"-Taste betätigen, so arbeitet der Prozessor den ersten, also den jetzt sichtbaren Befehl ab, anschließend übernimmt das Grundprogramm sofort wieder die Kontrolle, und es wird uns in gleicher Weise der folgende Befehl angezeigt. Also drücken wir die „CR"-Taste, und sogleich erscheint die *Abb. 3.1.14*. Daß in der Zwischenzeit der erste Befehl abgearbeitet wurde, war zeitlich nicht zu merken.

Im niederwertigen Teil des Registers D3 steht nun der Wert 3, im höherwertigen $FFFF. Unser „MOVE.W"-Befehl hatte ja auch nur die Aufgabe, eine 16-Bit-Größe zu laden, und die ist auch im Register angekommen.

Um den zweiten Befehl zu starten, drücken wir erneut die Taste „CR". Es erscheint die *Abb. 3.1.15*. Das ist die Situation nach dem zweiten Befehl, der dritte wird dabei bereits wieder angezeigt.

Im Register D0 ist der Wert $32 angekommen, das ist der sedezimale Wert von 50. Weiter gehts mit „CR" und somit mit dem Abarbeiten des nächsten Befehls.

Die *Abb. 3.1.16* zeigt die neue Situation. Der Befehl „JSR @SCHREITE" wurde ausgeführt, und deshalb erscheint die erste Seite unseres Quadrats auf dem Bildschirm.

Das Bild flimmert jetzt auch, denn die Schildkröte wird mit eingeblendet. Wenn man das abschalten will, so drückt man die Taste „S". Daraufhin erscheint links oben im Bildschirm ein Eingabefenster. Wir geben den Wert 0 ein. Daraufhin wird nur noch die Bildseite 0 angezeigt. Wenn man einfach „CR" eingibt, so wird die Bildumschaltung wieder aktiv. Die GDP64-Baugruppe besitzt nämlich vier Bildseiten, die unabhängig voneinander beschrieben und gelesen werden können. Die Schildkröte wird auf Bildseite 1 gezeichnet, und die Graphik auf Bildseite 0. Dann werden die beiden Seiten alle 20 Millisekunden umgeschaltet. Dadurch ergibt sich der Eindruck, daß beide Seiten quasi simultan sichtbar sind. Mit dem S-Befehl im Einzelschritt kann man eine Bildseite gezielt ansprechen, die dabei einzugebende Zahl von 0 bis 3 steht für die

```
D0       D1       D2       D3       D4       D5       D6       D7
FFFFFFFF 00000190 00000001 FFFF0024 0000000C 0000000D FFFFFFFF FFFFFFFE
A0       A1       A2       A3       A4       A5       A6       A7
00000A80 00008A4E 00008070 00008A60 FFFFFFFF 00008000 FFF7FFFF 0000BFF0
SR   USP      SSP      PC
A700 0000BFF0 0000BFF0 00009C00  move #4-1,d3          * vier Mal
```

Abb. 3.1.13
Im Einzelschritt erscheint die Textzeile mit eingeblendet, die als nächstes ausgeführt wird

```
D0       D1       D2       D3       D4       D5       D6       D7
FFFFFFFF 00000190 00000001 FFFF0003 0000000C 0000000D FFFFFFFF FFFFFFFE
A0       A1       A2       A3       A4       A5       A6       A7
00000A80 00008A4E 00008070 00008A60 FFFFFFFF 00008000 FFF7FFFF 0000BFF0
SR   USP      SSP      PC
A700 0000BFF0 0000BFF0 00009C04  move.w #50,d0         * Seitenlaenge
```

Abb. 3.1.14
Im Register D3.W befindet sich nun der Wert 4-1, also 3. Die höherwertigen 16 Bit sind undefiniert

```
D0       D1       D2       D3       D4       D5       D6       D7
FFFF0032 00000190 00000001 FFFF0003 0000000C 0000000D FFFFFFFF FFFFFFFE
A0       A1       A2       A3       A4       A5       A6       A7
00000A80 00008A4E 00008070 00008A60 FFFFFFFF 00008000 FFF7FFFF 0000BFF0
SR   USP      SSP      PC
A700 0000BFF0 0000BFF0 00009C08  jsr @schreite         * vorwaerts
```

Abb. 3.1.15
Nach dem nächsten „CR" muß eine Linie auf dem Bildschirm erscheinen

```
D0       D1       D2       D3       D4       D5       D6       D7
00000000 00000100 00000099 FFFF0003 0000000C 0000000D FFFFFFFF FFFFFFFE
A0       A1       A2       A3       A4       A5       A6       A7
00000A80 00008A4E 00008070 00008A60 FFFFFFFF 00008000 FFF7FFFF 0000BFF0
SR   USP      SSP      PC
2704 0000BFF0 0000BFF0 00009C0E  move.w #90,d0         * Winkelangabe
```

Abb. 3.1.16
Jetzt wird der Winkel für den Drehe-Befehl geladen

Abb. 3.1.17
Die Ziffer „1" soll gezeichnet werden

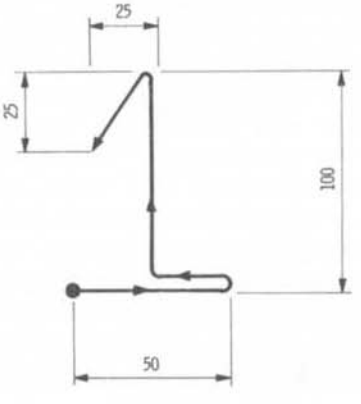


Abb. 3.1.18
Zerlegung in Elemente der Schildkrötenbewegung

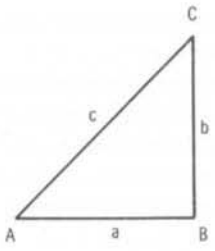


Abb. 3.1.19
Der Satz von Pythagoras

$$c^2 = a^2 + b^2$$

$$c = \sqrt{2} \cdot a$$

$$c = \frac{2 \cdot a}{\sqrt{2}}$$

Abb. 3.1.20
Berechnung der Seite „c"

```
ziffer1:
 move #-90,d0
 jsr @drehe
 move #50,d0
 jsr @schreite
 move #-25,d0
 jsr @schreite
 move #90,d0
 jsr @drehe
 move #100,d0
 jsr @schreite
 move #90+45,d0
 jsr @drehe
 move #5000/141,d0
 jsr @schreite
 rts
```

Abb. 3.1.21
Das fertige Programm

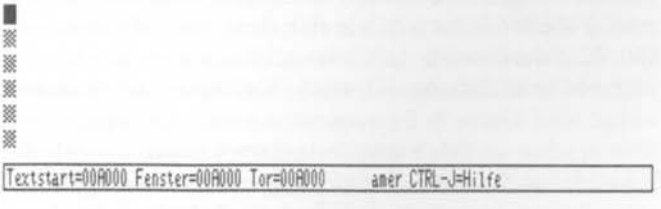


Abb. 3.1.22 So sieht die Ziffer auf dem Bildschirm aus

F=Flip M=Menue

angewählte Seite. Durch ein „CR" ohne Zahl wird die automatische Umschaltung wieder aktiv.

Jetzt aber zurück zu unserer Fehlersuche. Auf die bereits begonnene Weise, durch Drücken von „CR" jeweils einen Befehl auszuführen, können wir nun unser Programm schrittweise ausführen. Will man mehrere Schritte auf einmal ablaufen lassen, so gibt man „N" ein; im daraufhin erscheinenden Eingabefenster kann man dann die Anzahl der Schritte vorgeben. Oder man drückt „B", dann kann man eine Endadresse angeben, bis zu der das Programm laufen soll.

Sie sollten ruhig alle Möglichkeiten bis zum Ende des Programms durchprobieren. Dann sollte das fertige Quadrat auf dem Bildschirm zu sehen sein. Beobachten Sie dabei auch die Veränderung der Register, insbesondere des Schleifenzählers.

Von großen und kleinen Zahlen

Als neue Aufgabe nehmen wir uns vor, eine Ziffer auf den Bildschirm zu zeichnen. Auch dazu verwenden wir wieder unsere Zeichensprache.

Zunächst wollen wir nur eine Ziffer, nämlich die Ziffer „1" zeichnen. Die Schildkröte steht anfangs am Startpunkt und zeigt nach oben. Unsere Ziffer soll nach dem Ablauf unseres Programmes wie die in *Abb. 3.1.17* aussehen. Um diese Vorgabe in ein Programm umwandeln zu können, müssen wir die Zeichnung zunächst in die Elemente der Schildkrötenbewegung zerlegen. Die *Abb. 3.1.18* zeigt das Ergebnis dieser Überlegungen. Im Bild ist auch schon eine Bemaßung in Schritten eingetragen, so daß es leichter fällt, das Bild zu konstruieren.

Da die Schildkröte beim Start nach oben zeigt, muß sie zunächst eine Drehung nach rechts durchführen.

Dann werden 50 Schritte vorwärts durchgeführt, anschließend muß die Schildkröte wieder ein Stück zurück, um sich dann nach oben zu drehen. Dort angekommen, sind 100 Schritte vorwärts fällig, bevor sie sich dann nach links unten dreht. Schließlich muß sie noch solange schreiten, bis der eingezeichnete Endpunkt erreicht ist. Hier wartet eine kleine Schwierigkeit auf uns. Wir müssen festlegen, wieviele Schritte in diesem schrägen Stück erforderlich sind, haben aber nur eine Bemaßung der rechtwinkligen Linien. Für diese Berechnung nehmen wir uns den Satz des Pythagoras zu Hilfe (*Abb. 3.1.19*). In unserem Fall sind die Seiten a und b gleich lang, folglich ergibt sich für die gesuchte Seite c die Vereinfachung laut *Abb. 3.1.20*. Für den Rechner ist die Gleichung darunterstehend noch etwas umgeformt; anstatt die Seitenlänge a mit Wurzel 2 zu multiplizieren, wird sie mit 2 multipliziert und dann durch Wurzel 2 dividiert.

Die *Abb. 3.1.21* zeigt das Programm. In der ersten Zeile wird der Wert -90 in das Register D0 geladen. Der negative Wert bewirkt, daß sich die Schildkröte beim nachfolgenden Drehe-Befehl nicht nach links, sondern nach rechts dreht. Man hätte natürlich auch den Wert 270 laden können, die Schildkröte würde sich dann entsprechend weit links herum drehen und als Ergebnis in die gleiche Richtung zeigen. Zu dem folgenden Schreite-Befehl mit dem vorausgehenden Laden des Wertes ist nichts mehr zu sagen.

Doch zur Vorbereitung auf den wiederum darauf folgenden Schreite-Befehl laden wir erneut einen negativen Wert. Der Wert -25 veranlaßt hier die Schildkröte, rückwärts zu schreiten. Dadurch sparen wir uns ein Umkehren durch einen 180°-Drehe-Befehl. Danach wird die Schildkröte wieder nach oben ausgerichtet, und es werden die 100 Schritte nach oben ausgeführt. Jetzt kommt eine weitere Besonderheit. Wir wissen, daß sich die Schildkröte nach links unten drehen soll. Man kann die Berechnung des Winkels dem Grundprogramm überlassen. Wir wissen, man muß die Schildkröte um 90° nach links drehen, um sie horizontal auszurichten; wenn man sie dann noch um 45° dreht, so wird sie exakt entlang der Diagonale blicken. Also schreibt man im Assembler einfach „MOVE #90+45,D0". Damit wird der Wert 135 in das Register D0 geladen. Die Addition 90 + 45 wird also bereits bei der Übersetzung durch den Assembler vorgenommen, so daß sich der entstehende Maschinencode nicht mehr von der Angabe „MOVE #135,D0" unterscheidet. In diesem einfachen Beispiel ist der erkennbare Vorteil des Übergebens von Rechenaufgaben an den Assembler noch recht gering.

Bei dem nächsten Befehl ist es schon sinnvoller. Die Schildkröte soll entlang der Diagonalen laufen. Dazu muß gemäß unserer Formel die Seitenlänge mit 2 multipliziert und dann durch Wurzel 2 dividiert werden. Wurzel aus 2 ist aber 1.4142..., also eine Zahl mit Nachkommastellen. Das Grundprogramm kann aber nur mit ganzen Zahlen rechnen, daher muß man zu einem Trick greifen. Man multipliziert den Nenner und den Zähler mit einem großen Wert, rückt also das Komma nach rechts. Somit verschwinden die Nachkommastellen, und der Wert ist vom Grundprogramm in jetzt ausreichender Genauigkeit errechenbar.

Beispiel:

Wir müssen berechnen: 2 × 25 / 1.4142135

Man multipliziert Nenner und Zähler mit 100:

5000/141.2135

Nun kann man die Nachkommastellen weglassen:

5000/141

und das ist genau die Angabe in unserem Programm. Die Berechnung des Wertes überlassen wir dem Grundprogramm.

So, damit genug der Theorie, wir können das Programm eintippen, übersetzen und mit dem Namen „ZIFFER1" starten. Wenn sich keine Fehler eingeschlichen haben, so sollte nun auf dem Bildschirm die *Abb. 3.1.22* zu sehen sein.

Man könnte so für jede Ziffer einen Programmteil schreiben.

Nun wäre es natürlich interessant, verschieden große Ziffern auszugeben, ohne für jede Größe ein neues Programm zu schreiben. Dazu muß man das Programm unabhängig von der absoluten Größe der Ziffer gestalten. Wie geht das?

Wir suchen uns in unserem Programm die kleinste vorkommende Länge einer Linie. Diese Länge geben wir dann in einem Register vor. Alle anderen vorkommenden Linien werden im Programm dann nicht mehr durch angegebene feste Werte gezeichnet, sondern von diesem einen vorgegebenen Wert abgeleitet. Wenn wir nun die Größe ändern wollen, so ist nur noch ein Wert zu modifizieren. Sehen wir uns unser Programm dahingehend nochmal an. In unserer Ziffer „1" kommt die Länge 25 als kleinste Einheit vor. Also geben wir künftig nur noch diese Linie vor und leiten alle anderen davon ab.

```
ziffer1:
 move #60,d3 * aktuelle Groesse laden
 move #-90,d0
 jsr @drehe
 move d3,d0
 mulu #2,d0  * 2*Groesse
 jsr @schreite
 move d3,d0
 neg d0      * -Groesse
 jsr @schreite
 move #90,d0
 jsr @drehe
 move d3,d0
 mulu #4,d0  * 4*Groesse
 jsr @schreite
 move #90+45,d0
 jsr @drehe
 move d3,d0
 mulu #200,d0 * 2*Groesse/1.41
 divu #141,d0
 jsr @schreite
 rts
```

```
Textstart=00A000 Fenster=00A002 Tor=00A002 einf amer CTRL-J=Hilfe
```

Abb. 3.1.23 Dieses Programm kann Ziffern mit unterschiedlicher Größe erzeugen

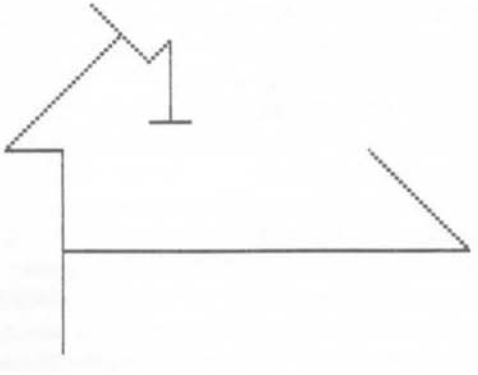


Abb. 3.1.25 So sieht der Bildschirm nach dem Start des Programms aus Abb. 3.1.24 aus

F=Flip M=Menue

```
Rolf-D.Klein 68000/08 Assembler 4.3 (C) 1984, Seite 1


009C00
009C00                  *
009C00                  * PROGRAMM FUER UNTERSCHIEDLICH
009C00                  * GROSSE ZIFFERN
009C00                  *
009C00
009C00
009C00                  ZIFFER:       * SCHRIFTGROESSE IN D3
009C00  303C FFA6        MOVE #-90,D0 *
009C04  4EB9 000012F6    JSR @DREHE
009C0A  3003             MOVE D3,D0
009C0C  C0FC 0002        MULU #2,D0    * 2*GROESSE
009C10  4EB9 00001288    JSR @SCHREITE
009C16  3003             MOVE D3,D0
009C18  4440             NEG D0
009C1A  4EB9 00001288    JSR @SCHREITE
009C20  303C 005A        MOVE #90,D0
009C24  4EB9 000012F6    JSR @DREHE
009C2A  3003             MOVE D3,D0
009C2C  C0FC 0004        MULU #4,D0    * 4*GROESSE
009C30  4EB9 00001288    JSR @SCHREITE
009C36  303C 0087        MOVE #90+45,D0
009C3A  4EB9 000012F6    JSR @DREHE
009C40  3003             MOVE D3,D0
009C42  C0FC 00CB        MULU #200,D0 * 2*GROESSE/1.41
009C46  80FC 008D        DIVU #141,D0
009C4A  4EB9 00001288    JSR @SCHREITE
009C50  4E75             RTS
009C52
009C52
009C52                  START:
009C52  363C 000A        MOVE #10,D3   * SCHRIFTGROESSE
009C56  6100 FFA8        BSR ZIFFER    * UNTERPROGRAMMAUFRUF
009C5A  363C 0014        MOVE #20,D3   * SCHRIFTGROESSE
009C5E  6100 FFA0        BSR ZIFFER    * UNTERPROGRAMMAUFRUF
009C62  363C 0032        MOVE #50,D3   * SCHRIFTGROESSE
009C66  6100 FF98        BSR ZIFFER    * UNTERPROGRAMMAUFRUF
009C6A  4E75             RTS           * ENDE DES HAUPTPROGRAMMS
009C6C
009C6C
009C6C
009C6C
009C6C
009C6C

008AB4  Ende-Symboltabelle
0093B4  Ende-Debug-Tabelle
```

Abb. 3.1.24 Das Programmlisting zum universellen Ziffern-Ausgabeprogramm; „START" ist der Programmanfang

Die *Abb. 3.1.23* zeigt das neue Programm.

In der ersten Programmzeile, nach der Namensdefinition von „ZIFFER1", wird der Wert 60 in das Register D3 geladen. Dies soll die neue Größe der kleinsten Zeicheneinheit sein.

Nach der Drehung wird diesmal nicht der Wert 50 geladen, sondern mit „MOVE D3,D0" der Inhalt des Registers D3 nach D0. D3 enthält ja jetzt die Größe. Mit „MULU #2,D0" wird dieser Wert mit 2 multipliziert.

Damit steht in D0 der doppelte Wert der eingegebenen Größe.

Um diesen Wert wird vorwärts geschritten. Dann wird wieder D3 nach D0 geladen. Diesmal wird mit „NEG D0" der Wert negiert, also das Vorzeichen umgekehrt, denn jetzt soll die Schildkröte wieder zurück und auf halber Strecke stehenbleiben.

Nach der Drehung um 90° wird dann erneut multipliziert, diesmal mit vier. Anschließend legt die Schildkröte also den vierfachen Weg unserer Elementarstrecke, also der ursprünglich eingegebenen, zurück. In unserem ersten Beispiel mit den festen Werten stand hier der Wert 100. Nach der Drehung um 135° folgt nun die komplizierte Diagonale. Dazu wird die Elementarlänge aus D3 mit 200 multipliziert und mit dem Befehl „DIVU #141,D0" durch 141 dividiert. Es wird hier also die gleiche Formel wie vorher angewendet, nur daß sie diesmal erst bei der Ausführung des Programms berechnet werden kann und nicht schon vorab durch das Grundprogramm oder den Programmierer.

Die Assemblerbefehle MULU, DIVU und NEG

In diesem Programm haben wir drei neue Befehle des 68000/8 kennengelernt. „MULU" ist die Abkürzung für „multiply unsigned", oder Mulipliziere ohne Vorzeichen. „DIVU" steht für „divide unsigned", also Dividiere ohne Vorzeichen.

Will man jetzt verschieden große Ziffern ausgeben, so muß man nur den Lade-Befehl am Anfang des Programms ändern. Die Ziffer bleibt in richtiger Proportion erhalten. Wenn man den Ladebefehl wegläßt, so erhält man einen Programmteil, der eine Ziffer in Abhängigkeit von dem Registerinhalt D3 unterschiedlich groß ausgibt.

Nun sollen unterschiedlich große Ziffern auf den Bildschirm gezeichnet werden.

Die *Abb. 3.1.24* zeigt das Programm. Diesmal ist nicht der Inhalt des Editor-Fensters abgedruckt, sondern das Assemblerlisting eines Druckers. Wenn Sie das Programm eingeben, so tippen Sie nur die rechte Hälfte ein, nicht die Zahlencodes der linken Seite. Die werden bei der Übersetzung vom Assembler erzeugt.

Es sind jetzt praktisch zwei Programme dargestellt. Das erste mit dem Namen „ZIFFER" kennen Sie schon, es ist fast identisch mit dem aus der Abb 3.1.23. Am Anfang fehlt das Laden des Registers D3, und der Name wurde geändert.

Dahinter, also nach RTS, folgt ein neuer Programmteil. Er beginnt mit „START:". Dies ist das sogenannte Hauptprogramm, während das erste Programm ein sogenanntes Unterprogramm war.

Der Assemblerbefehl BSR

Im Hauptprogramm gibt es nun einen Befehl, den Sie noch nicht kennen: „BSR". „BSR" bedeutet „branch subroutine", auf deutsch, „springe ins Unterprogramm". Damit ist es also möglich, wie mit dem „JSR„-Befehl, vorgefertigte Unterprogramme zu verwenden. Dahinter steht der Name des Unterprogramms, in unserem Fall der Name „ZIFFER".

Anstelle des „BSR„-Befehls hätte man auch den „JSR„-Befehl verwenden können. Die beiden Befehle haben einen Unterschied, der in der Art der Adressierung liegt. Während der „BSR„-Befehl relativ adressiert, adressiert der „JSR„-Befehl absolut. Relativ bedeutet, daß der Adreßteil, der das Sprungziel bestimmt, relativ zur aktuellen Speicheradresse genommen wird, also einen Abstand zur aktuellen Position bedeutet. Absolut dagegen besagt, daß die Sprungzieladresse direkt eingesetzt wird. In der Abb. 3.1.24 wird das deutlich. Beim „JSR @SCHREITE„-Befehl steht der Objektcode „4EB9 00001288", die Adresse 00001288 ist die Adresse im Grundprogramm, wo das Unterprogramm SCHREITE beginnt. Beim Befehl „BSR ZIFFER" stehen jeweils unterschiedliche Objektcodes, beim ersten z. B. „6100 FFAB", beim nächsten Aufruf steht „6100 FFA0", denn der Abstand zum Unterprogramm „ZIFFER" ist gewachsen, die Darstellung der Distanz erfolgt im Zweierkomplement.

Der „BSR"-Befehl weist folgende Vorteile auf. Zum einen benötigt er weniger Bytes, da nicht die gesamte Adresse, sondern nur ein Abstand angegeben werden muß. Zum anderen könnten wir unser Gesamtprogramm, also Unterprogramm plus Hauptprogramm, an eine andere Speicherstelle verschieben, ohne am Programm etwas ändern zu müssen. Die Abstände innerhalb des Programms sind ja gleichgeblieben, während sich dabei die absoluten Adressen natürlich ändern würden.

Als Nachteil des „BSR„-Befehles ist der geringe adressierbare Bereich zu nennen. Da die Angabe der Entfernung auf 16 Bits begrenzt ist, kann man nur einen Bereich von ca. +/-32767 Bytes anspringen. Aus diesem Grund eignet er sich auch nicht zum Aufruf der Grundprogramm-Unterprogramme, denn die sind meistens weiter entfernt.

Doch jetzt weiter in unserer Programmbeschreibung. Vor jedem Aufruf wird das Register D3 mit einem Wert belegt, nämlich der Größe der Ziffer. Diese soll ja in unterschiedlichen Größen ausgegeben werden. Der Programmteil „START" wird auch mit einem „RTS" abgeschlossen. „RTS" ist die Abkürzung für „return from subroutine" also „Rückkehr aus dem Unterprogramm".

Die *Abb. 3.1.25* zeigt das Ergebnis, nachdem man das Programm eintippt, übersetzt und startet. Dabei muß es mit dem Namen „START" aufgerufen werden.

So haben Sie sich die Ziffern sicher nicht vorgestellt. Wir haben noch einen Fehler im Programm. Allerdings wurde ja auch nicht gesagt, wie die Ziffern auf dem Bildschirm erscheinen sollen.
Sie sollen senkrecht stehen. Dazu muß man im Unterprogramm „ZIFFER" dafür sorgen, daß die Schildkröte nach dem Zeichnen einer Ziffer wieder an ihren Ausgangsort zurückkehrt. Die *Abb. 3.1.26* zeigt eine mögliche Lösung.

Am Schluß des Programmteils „ZIFFER" wurden ein paar Befehle eingefügt.

Neu sind die Befehle „@HEBE" und „@SENKE". Wie Sie sicher bereits an der Schreibweise, bzw. dem Aufruf bemerkt haben, handelt es sich dabei natürlich nicht um Befehle des 68000/8, sondern um Grundprogramm-Befehle, also um Unterpro-

```
Rolf-D.Klein 68000/08 Assembler 4.3 (C) 1984, Seite 1

009C00
009C00                    *
009C00                    * PROGRAMM FUER UNTERSCHIEDLICH
009C00                    * GROSSE ZIFFERN
009C00                    *
009C00
009C00
009C00                    ZIFFER:       * SCHRIFTGROESSE IN D3
009C00  303C FFA6          MOVE #-90,D0 *
009C04  4EB9 000012F6      JSR @DREHE
009C0A  3003               MOVE D3,D0
009C0C  C0FC 0002          MULU #2,D0    * 2*GROESSE
009C10  4EB9 00001288      JSR @SCHREITE
009C16  3003               MOVE D3,D0
009C18  4440               NEG D0
009C1A  4EB9 00001288      JSR @SCHREITE
009C20  303C 005A          MOVE #90,D0
009C24  4EB9 000012F6      JSR @DREHE
009C2A  3003               MOVE D3,D0
009C2C  C0FC 0004          MULU #4,D0    * 4*GROESSE
009C30  4EB9 00001288      JSR @SCHREITE
009C36  303C 0087          MOVE #90+45,D0
009C3A  4EB9 000012F6      JSR @DREHE
009C40  3003               MOVE D3,D0
009C42  C0FC 00C8          MULU #200,D0 * 2*GROESSE/1.41
009C46  B0FC 008D          DIVU #141,D0
009C4A  4EB9 00001288      JSR @SCHREITE
009C50  303C FF79          MOVE #-90-45,D0 * ZURUECKDREHEN
009C54  4EB9 000012F6      JSR @DREHE
009C5A  4EB9 0000108A      JSR @HEBE     * KEINE LINIE ZEICHNEN
009C60  3003               MOVE D3,D0
009C62  C0FC 0003          MULU #3,D0    * 4*GROESSE IST GESAMTHOEHE
009C66  4440               NEG D0        * 3*GROESSE IST DIFFERENZ
009C68  4EB9 00001288      JSR @SCHREITE
009C6E  4EB9 00001090      JSR @SENKE
009C74  4E75               RTS
009C76
009C76
009C76                    START:
009C76  363C 000A          MOVE #10,D3   * SCHRIFTGROESSE
009C7A  6100 FF84          BSR ZIFFER    * UNTERPROGRAMMAUFRUF
009C7E  363C 0014          MOVE #20,D3   * SCHRIFTGROESSE
009C82  6100 FF7C          BSR ZIFFER    * UNTERPROGRAMMAUFRUF
009C86  363C 0032          MOVE #50,D3   * SCHRIFTGROESSE
009C8A  6100 FF74          BSR ZIFFER    * UNTERPROGRAMMAUFRUF
009C8E  4E75               RTS           * ENDE DES HAUPTPROGRAMMS
009C90
009C90
009C90
009C90
009C90
009C90

008A72  Ende-Symboltabelle
0094CA  Ende-Debug-Tabelle
```

Abb. 3.1.26 Das erweiterte Programm „ZIFFER"

gramme im Grundprogramm. Damit kann man erreichen, daß die Schildkröte keine Schreibspur hinterläßt, während sie zum Ausgangsort zurückläuft. Also den an ihr montierten Schreibstift „hebt", oder „senkt".

Die *Abb. 3.1.27* zeigt das Ergebnis des Programmlaufs.

Aufgaben:

1. Verändern Sie das Hauptprogramm so, daß die einzelnen Ziffern nebeneinander stehen und sich nicht mehr berühren.
2. Programmieren Sie das Unterprogramm ZIFFER neu, so daß die Ziffer 2 dargestellt wird.

Aufteilung von Problemen

Sie haben die Aufgabe, ein einfaches Haus, wie es *Abb. 3.1.28* zeigt, mit der Schildkrötengraphik zu zeichnen. Bitte versuchen Sie es einmal, ohne die Lösung vorher anzusehen.

Wahrscheinlich haben Sie versucht, alles in einem Programm unterzubringen. Wie Sie sicher bemerkt haben, entsteht dabei ein sehr langes und unübersichtliches Programm. Einfacher und übersichtlicher ist es, ein kompliziertes Problem erst in Teilprobleme zu zerlegen. In der Haus-Figur erkennt man zwei Bilder, die sich einfach programmieren lassen. Ein Quadrat und ein Dreieck. Das Quadrat ist schon vom Anfang des Kapitels her bekannt, und das Dreieck läßt sich daraus einfach ableiten.

Ein Dreieck besteht aus drei Seiten, also braucht man ein Programmstück, das in einer Schleife dreimal durchlaufen wird. Damit sich die Schildkröte zum Schluß wieder in der Ausgangslage befindet, muß sie sich um einen Gesamtwinkel von 360° drehen. Wir wollen diese Drehung in drei Teilen absolvieren, darum teilt man den Gesamtwinkel durch 3 und erhält 120° für die Drehung nach jeder Seitenlinie.

Zunächst haben wir also unsere Aufgabe in zwei Teilaufgaben zerlegt. Wir benötigen also ein Unterprogramm „QUADRAT" und ein Unterprogramm „DREIECK", die wir dann von unserem Hauptprogramm „HAUS" aus verwenden können. Das Hauptprogramm ruft dann einfach die beiden Unterprogramme auf. *Abb. 3.1.29* zeigt das fertige Programm.

Tippen Sie das Programm ein, übersetzen Sie es mit dem Assembler und starten es ab der Adresse „HAUS".

Die *Abb. 3.1.30* zeigt das Ergebnis. Offensichtlich haben wir einen Fehler gemacht.

Aufgaben:

1. Versuchen Sie, den Fehler im Programm „HAUS" zu finden, indem Sie das Programm im Einzelschritt durchlaufen. Dazu muß der DEBUG-MODE mit Hilfe des Optionen-Menues eingeschaltet werden, das Programm erneut übersetzt und im Einzelschritt-Menue durchlaufen werden.

Abb. 3.1.27 Das Ergebnis des Programms aus Abb. 3.1.26

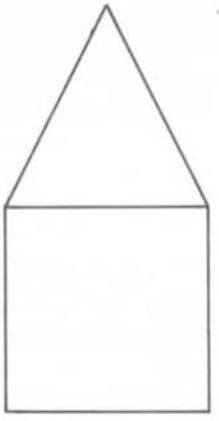
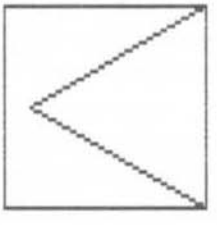


Abb. 3.1.28 Schreiben Sie ein Programm, das ein Haus zeichnet; die absolute Größe und die Seitenverhältnisse spielen dabei keine Rolle

```
quadrat:
 move #4-1,d3
quad1:
 move #100,d0
 jsr @schreite
 move #90,d0
 jsr @drehe
 dbra d3,quad1:
 rts

dreieck:
 move #3-1,d3
drei1:
 move #100,d0
 jsr @schreite
 move #120,d0
 jsr @drehe
 dbra d3,drei1
 rts

haus:
 bsr quadrat
 bsr dreieck
 rts
```

```
Textstart=009000 Fenster=009000 Tor=00900E      amer CTRL-J=Hilfe
```

Abb. 3.1.29 Ein erster Versuch das Programm „haus" zu schreiben



```
F=Flip M=Menue
```

Abb. 3.1.30 Das Programm „haus" liefert kein Haus

2. Wenn Sie den Fehler entdeckt haben, versuchen Sie, ihn zu korrigieren.

Hinweis: Im Einzelschritt werden hier auch alle Unterprogramme schrittweise und nicht wie bei unseren ersten Versuchen mit dem Einzelschrittmodus als ein Befehl ausgeführt. Wenn Sie das nicht wollen, so müssen die „BSR„-Aufrufe durch „JSR„-Aufrufe ersetzt werden.

Vielleicht haben Sie eine andere Lösung gefunden, als die in *Abb. 3.1.31* gezeigte; bei Programmen gibt es meist mehrere Lösungen, die zum Ziel führen. Auf jeden Fall sollte das Ergebnis der *Abb. 3.1.32* entsprechen.

Wie schön es ist, mit Unterprogrammen zu arbeiten, zeigt eine weitere Aufgabenstellung, die wir aus unserem Programm „HAUS" ableiten können. Nehmen wir uns vor, mehrere Häuser auf den Bildschirm zu bringen.

Wenn man nämlich statt einem Haus viele haben will, so verwendet man nun das Programm „HAUS" als weiteres Unterprogramm. Das Hauptprogramm lautet nun „HAEUSER". Dabei sollen mehrere Häuser an unterschiedlichen Stellen des Bildschirms erscheinen. Eine Möglichkeit besteht nun darin, zwischen den einzelnen Häusern mit den Befehlen „HEBE" und „SENKE" den Schreibstift der Schildkröte zu steuern und mit entsprechenden Befehlen den neuen Anfangspunkt des Hauses anzufahren. Doch daraus ergibt sich eine recht umfangreiche Programmierung für die Bewegung der Schildkröte, ohne daß wir eigentlich eine Grafik dadurch erhalten. Darum gibt es für solche Fälle einen weiteren Befehl bzw. ein weiteres Unterprogramm in unserem Grundprogramm: „JSR @SET". Mit dem Unterprogramm „@SET" kann man die Position der Schildkröte in absoluten Koordinaten einstellen. Bisher wurde die Schildkröte mit „SCHREITE" und „DREHE" relativ zum Ausgangsort bewegt. Bei „SET" gibt man die Endposition direkt an.

Das Unterprogramm „@SET" verlangt drei Werte in verschiedenen Registern. Im Register D1 steht die X-Koordinate. Wenn der Inhalt des Registers D1 gleich 0 ist, so liegt die Position ganz links, wenn der Inhalt von D1 gleich 511 ist, so wird die Schildkröte ganz rechts auf den Bildschirm positioniert. Im Register D2 steht die Y-Koordinate. Dabei ist 0 ganz unten und 511 ganz oben.

Hier noch ein wichtiger Hinweis: Der Bildschirm hat nur eine Auflösung von 512 mal 256 Punkten. Bei den Schildkröten-Koordinaten gibt man in Y-Richtung trotzdem als Maximalwert 511 an, obwohl eigentlich nur die Position 255 verwendet wird. Die Umrechnung geschieht im Grundprogramm, und man hat den Vorteil, daß man eine scheinbar symmetrische Auflösung in beiden Richtungen hat.

Im Register D3 wird der Winkel angegeben. 0° läßt dabei die Schildkröte nach rechts und 90° nach oben zeigen.

Aufgaben:

1. Versuchen Sie, das Bild mit mehreren Häusern zunächst ohne den „SET„-Befehl zu erzeugen. Hinweis: Sie benötigen dazu die Befehle „HEBE" und „SENKE".
2. „Vereinfachen" Sie Ihr Programm durch den neuen Befehl „SET".

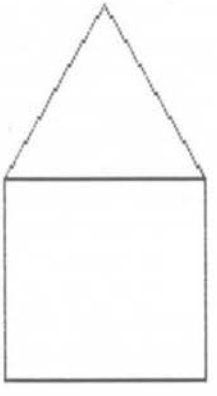
Kreise

Jetzt stellen wir uns die Aufgabe, einen Kreis zu zeichnen.

```
haus:
 bsr quadrat
 move #100,d0
 jsr @schreite
 move #30,d0
 jsr @drehe
 bsr dreieck
 rts
```

Abb. 3.1.31 Der Programmteil „haus" muß so korrigiert werden

```
Textstart=009000 Fenster=0090EF Tor=0090EF      amer CTRL-J=Hilfe
```



F=Flip M=Menue

Abb. 3.1.32 Der Bildschirm mit dem korrekten Bild des „haus"-Programms

```
kreis:
 move #1,d0
 jsr @schreite
 move #1,d0
 jsr @drehe
 bra kreis
```

Links: Abb. 3.1.33 Ein Kreisprogramm

```
Textstart=009000 Fenster=009000 Tor=009000      amer CTRL-J=Hilfe
```

```
kreis:
 move #360-1,d3
schleife:
 move #1,d0
 jsr @schreite
 move #1,d0
 jsr @drehe
 dbra d3,schleife
 rts
```

```
Textstart=009000 Fenster=009000 Tor=009000      amer CTRL-J=Hilfe
```

Abb. 3.1.34 Dieses Kreisprogramm hört nach Zeichnen des Kreises auf und das Menue meldet sich wieder

```
kreis:
 move #57,d0     * Radius einzeichnen
 jsr @schreite
 move #90,d0
 jsr @drehe      * Kreis drehen
 move #360-1,d3
schleife:
 move #1,d0
 jsr @schreite
 move #1,d0
 jsr @drehe
 dbra d3,schleife
 rts
```

```
Textstart=009000 Fenster=009000 Tor=009000      amer CTRL-J=Hilfe
```

Abb. 3.1.35 Kreisprogramm mit Radius-Eintrag

Mit der Schildkrötengraphik kann man das ganz leicht tun. Man läßt die Schildkröte ein kleines Stückchen schreiten, dann dreht man sie um einen kleinen Winkel, läßt sie wieder schreiten usw. usw.

Leider hat die Sache auch einen Haken. Man erhält keinen ganz runden Kreis, sondern einen eckigen, also eigentlich nur ein Vieleck. Auf dem Bildschirm hat man aber sowieso nur ein vorgegebenes Punktraster, und einen wirklich runden Kreis kann man also ohnehin nie zeichnen.

Die *Abb. 3.1.33* zeigt das Programm. Als kleinster Schritt wurde der Wert 1 verwendet, sowie als kleinster Winkel 1°.

Der Assemblerbefehl BRA

Der Befehl „BRA" ist neu. „BRA" bedeutet „branch", also „springe". Dieser Befehl veranlaßt den Prozessor, nicht an der Adresse des nächsten Befehls seine Arbeit fortzusetzen, sondern an der dahinter angegebenen Stelle. Es wird also gewissermaßen ein Sprung ausgeführt. Dahinter steht der Name „KREIS", also springt das Programm zur Marke „KREIS". Die Marke, also das Symbol „KREIS", muß also irgendwo im Programm bereits definiert worden sein. Der Assembler ersetzt diese Marke dann bei der Übersetzung durch den richtigen Adreßwert. Nachdem in unserem Fall „KREIS" weiter oben liegt, wird die Ausführung wiederholt. Somit schreitet die Schildkröte einen Schritt, dann dreht sie sich um ein Grad, dann schreitet sie wieder usw.

Wenn man das Programm mit dem Namen „KREIS" startet, so erscheint auf dem Bildschirm ein Kreis. Das Programm endet aber nicht mehr, da die Abarbeitung der Befehle durch den BRA-Befehl immer wieder von vorne beginnt. Wir haben also eine Endlosschleife pogrammiert. Will man das Programm stoppen, so muß man die RESET-Taste an der CPU-Baugruppe drücken.

Die erste Verfeinerung des Programms wäre also ein definierter Abbruch. Die Frage lautet: Wie oft muß die Schleife ausgeführt werden, damit ein Kreis auf den Bildschirm gezeichnet wird?

Dazu kennen wir schon den Schildkrötensatz, der lautet: Die Schildkröte muß sich insgesamt um 360° drehen, um wieder in die Ausgangsrichtung zu blicken.

Also muß der Gesamtwinkel der Drehung 360° betragen. Da wir die Schildkröte um je 1° pro Schleifendurchlauf drehen, muß die Schleife 360 mal durchlaufen werden, um den Kreis darzustellen.

Die *Abb. 3.1.34* zeigt das Programm. Es liefert ebenfalls einen Kreis, jedoch erscheint unten rechts die Meldung „F=Flip M=Menue", da das Programm auch ordnungsgemäß beendet wird.

Wie können wir nun bei Bedarf kleinere Kreise zeichnen? Einfach dadurch, daß wir die Schildkröte um mehr als 1° drehen. Als Beispiel nehmen wir 10° an. Natürlich muß man die Schleife dann nur noch 36 mal durchlaufen. Sehen wir uns also unser Programm dahingehend an. Wir müssen die Anzahl der Schleifendurchläufe ändern. Dazu laden wir in der zweiten Programmzeile nicht 360-1, sondern 36-1 in das Register D3. Außerdem, wie wir soeben festgestellt haben, wollen wir unsere Schildkröte um 10°

anstelle bisher 1° drehen. Wir müssen also bei der Vorbereitung des Drehe-Befehls anstelle einer „1" jetzt „10" in das Register D0 laden. Als Ergebnis erhalten wir einen kleineren Kreis.

Schön wäre es, jetzt Kreise mit vorgegebenem Radius zeichnen zu lassen. Dazu brauchen wir etwas Mathematik.

Da gibt es zunächst die Formel:

Umfang = $2 \times \pi \times$ Radius

π ist die Zahl 3.141592...., die für die Berechnung gebraucht wird.

Den Umfang unseres Kreises kann man berechnen, denn wir schreiten einen Schritt, und im ersten Beispiel wird das 360 mal durchgeführt. Also beträgt der Umfang $360 \times 1 = 360$ Schritte.

Der Radius war daher: $360 / (2 \times \pi) = 57.295...$

Also ca. 57 Bildpunkte.

Das kann man ausprobieren; dazu gibt man das Programm aus der *Abb. 3.1.35* ein. Es ergibt sich beim Ablauf des Programmes ein Kreis mit eingezeichnetem Radius.

Wir wollten aber eigentlich das umgekehrte Problem lösen. In unserem Programm gibt es zwei Freiheitsgrade, also zwei Werte, die wir verändern können. Zum einen können wir den Winkel verändern, um kleinere Kreise zu erhalten, zum anderen können wir die Schrittzahl erhöhen, um größere Kreise auf den Bildschirm zu bringen. Wir müssen uns also entscheiden, welche Größe berechnet werden soll. Falls wir uns für den Winkel entscheiden, so werden wir mit folgendem Problem konfrontiert: Wir können nur ganze Zahlen eingeben, und der Gesamtwinkel nach dem Schleifendurchlauf muß exakt 360° betragen. Es kommen darum nur Winkelwerte in Frage, durch die 360 ohne Rest teilbar ist, also 1,2,3,4,5,6,8,9,10,12,15,18,20,24,30,40,45,60,72,90,120, 180. Damit lassen sich aber nicht alle vorgegebenen Radien bestimmen.

Führen wir uns unsere Gedanken von vorhin nochmals vor Augen. Wir stellten fest, daß wir (ausgehend von Einerschritten bei Winkel und Schrittweite) durch Vergrößern des Winkels kleinere, durch Vergrößern der Schrittweite aber größere Kreise erhalten. Wenn wir nun einen größeren Winkel als Konstante wählen, so können wir durch Variieren der Schrittweite die Größe unseres Kreises in einem weiten Bereich verändern. Die Formel lautet dann:

Umfang = $36 \times$ Schrittweite

Umfang = $2 \times \pi \times$ Radius

also: Schrittweite = $(2 \times \pi \times R) / 36$

Wir wählen mal einen Radius von 100. Damit ergibt sich:

Schrittweite = $(2 \times + \pi \times 100) / 36$

Schrittweite = 17.45....

Nachdem nur ganze Schritte ausführbar sind, verwendet man 17. Die fehlenden Nachkommstellen führen natürlich zu einer Ungenauigkeit, denn der wahre Radius beträgt nun:

$(36 \times 17)/(2 \times \pi) = 97.40$ ...

Damit ergibt sich eine Differenz von ca. 3. Besser wäre ein SCHREITE-Befehl, der nicht nur Einer-Schritte ausführen, sondern auch genauer arbeiten kann. Tatsächlich

```
kreis:
 move #100,d3    * Radius vorgeben
 mulu #279,d3    * (2*PI*Radius)/36*16
 divu #100,d3    * in gerundeter Form
 move #36-1,d4   * Anzahl der Durchlaeufe
schleife:        *
 move d3,d0
 jsr @schr16tel * mit brechneter Zahl
 move #10,d0     * und 10 Grad
 jsr @drehe
 dbra d4,schleife
 rts
```

Abb. 3.1.36 Programm für beliebige Radien

```
Textstart=009000 Fenster=009000 Tor=009000      aner CTRL-J=Hilfe
```

```
kreis:
 move #100,d3    * Radius vorgeben
 mulu #279,d3    * (2*PI*Radius)/36*16
 divu #100,d3    * in gerundeter Form
 move #36-1,d4   * Anzahl der Durchlaeufe
schleife:        *
 move #5,d0      * zuerst Drehen
 jsr @drehe
 move d3,d0
 jsr @schr16tel * mit brechneter Zahl
 move #5,d0      * und gesamt 10 Grad
 jsr @drehe      * Drehung, aber
 dbra d4,schleife * symmetrisch.
 rts
```

Abb. 3.1.37 Programm für symmetrische Kreise mit beliebigen Radien

```
Textstart=009000 Fenster=009000 Tor=009000      aner CTRL-J=Hilfe
```

gibt es einen genaueren SCHREITE-Befehl, den SCHR16TEL-Befehl, der um 1/16 Punkt schreiten kann. Denn intern, im Grundprogramm, wird die Position der Schildkröte sehr viel genauer gespeichert, als dies auf dem Bildschirm sichtbar ist.

Mit dem SCHR16TEL-Befehl kann man dann jeden Radius vorgeben.

Die neue Formel lautet dann (auch bei 10° Drehung mit 36 Schleifendurchläufen):

Umfang = 36 × Schrittweite16tel / 16

Umfang = 2 × π × Radius

also:

Schrittweite16tel = (2 × π × Radius) / 36 × 16

Durch die Multiplikation mit 16 am Schluß der Formel ergibt sich ein größerer Wert für die neue Schrittweite. Das Weglassen der Nachkommastellen verursacht also einen deutlich kleineren Fehler.

Beispiel: Der Radius soll 100 betragen. Damit ergibt sich:

Schrittweite16tel = 279.25...

Wenn wir zur Probe 279 als Schrittweite16tel wählen, so ergibt sich umgekehrt ein Radius von 99.909... Die Abweichung liegt also in einer fast nicht meßbaren Dimension, denn auf dem Bildschirm werden nur ganze Punkte sichtbar. 99.9 entspricht auf dem Bildschirm 100 Bildpunkten, somit ist keine Abweichung mehr sichtbar.

Genug der Theorie. Nun ein praktisches Programm, bei dem der Radius im Register D3 vorgegeben werden kann, und das dann den entsprechenden Kreis zeichnet.

Die *Abb. 3.1.36* zeigt das fertige Programm. In der ersten Zeile wird der Radius in das Register D3 geladen. Dann wird der Inhalt mit 2.79 multipliziert. Dies geschieht dadurch, daß man den Wert zunächst mit 279 multipliziert und anschließend durch 100 teilt.

Das Programm führt im wesentlichen all die Berechnungen durch, die wir soeben besprochen haben; es hat aber noch einen kleinen Schönheitsfehler, der aber nicht im Listing, sondern nur im Betrieb auf dem Bildschirm zu sehen ist. Die Schildkröte bleibt nicht ganz exakt symmetrisch zum Kreis stehen. Das kommt daher, daß der Kreis nicht symmetrisch zur Mittelachse gezeichnet wird, denn wir schreiten zuerst, dann wird gedreht. Um das zu verstehen, zeigt die *Abb. 3.1.37* eine verbesserte Version des Programms. Wenn Sie die beiden Listings miteinander vergleichen, so werden Sie erkennen, daß im verbesserten Programm der Schreite16tel-Befehl zwischen zwei Teildrehungen von je 5° liegt. Jetzt arbeitet unser Zeichenprogramm also symmetrisch. Dieses Programm können Sie jetzt für alle möglichen Kreise verwenden.

Ein kleines Testprogramm zeigt, was sich nun mit diesem Programm in Verbindung mit einer kleinen Erweiterung anstellen läßt. Die *Abb. 3.1.38* zeigt das Listing, und in der *Abb. 3.1.39* ist das Ergebnis zu sehen.

```
test:
 move #200,d5    * Radius vorgeben
kreis:
 move d5,d3      * Radius holen
 mulu #279,d3    * (2*PI*Radius)/36*16
 divu #100,d3    * in gerundeter Form
 move #36-1,d4   * Anzahl der Durchlaeufe
schleife:        *
 move #5,d0      * zuerst Drehen
 jsr @drehe
 move d3,d0
 jsr @schr16tel * mit brechneter Zahl
 move #5,d0      * und gesamt 10 Grad
 jsr @drehe      * Drehung, aber
 dbra d4,schleife * symmetrisch.
 dbra d5,kreis
 rts
```

Abb. 3.1.38 Ein kleines Testprogramm

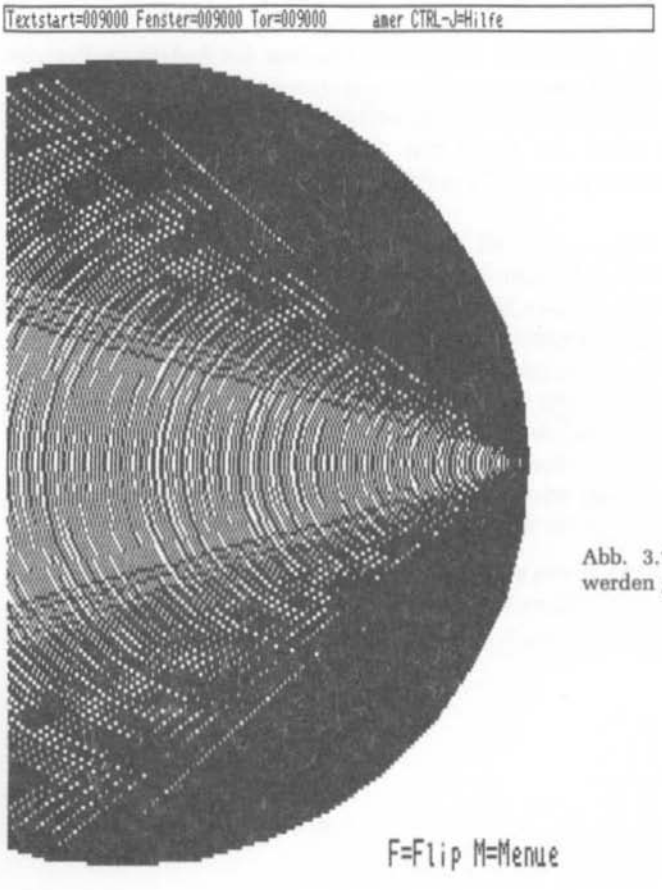


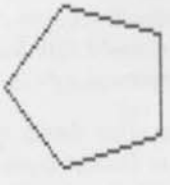
Abb. 3.1.39 Unterschiedliche Kreise werden gezeichnet

```
start:      *
 move #5,d3 * 5-Eck
poly:       * n-Eck in d3
 move #360,d4
 divu d3,d4 * Winkel in d4
 sub #1,d3  * fuer DBRA-Befehl
polschleife:
 move #50,d0
 jsr @schreite
 move d4,d0
 jsr @drehe
 dbra d3,polschleife
 rts
```

Abb. 3.1.40 Ein 5-Eck-Programm

```
Textstart=009000 Fenster=009000 Tor=009000      aner CTRL-J=Hilfe
```

```
newpoly:
 move #125,d3    * Startwinkel
 move #200,d4    * Gesamtzahl
schleife:
 move #20,d0
 jsr @schreite
 move d3,d0
 jsr @drehe
 move #20,d0
 jsr @schreite
 move d3,d0
 add d0,d0       * Winkel * 2
 jsr @drehe
 dbra d4,schleife
 rts
```



F=Flip M=Menue

Abb. 3.1.41 Das 5-Eck auf dem Bildschirm

Links: Abb. 3.1.42 Ein Polygon-Programm

```
Textstart=009000 Fenster=009000 Tor=009000      aner CTRL-J=Hilfe
```

Polygone

Unser Kreis war eigentlich schon ein Polygon, nämlich ein 36-Eck.

Wie entsteht nun ein solches Polygon, oder zu deutsch Vieleck? Wir haben bereits festgestellt, daß unsere Kreise nichts anderes waren. Bei einem Polygon ändert sich aber natürlich das Verhältnis zwischen Drehen und Schreiten. Ein brauchbares Polygon-Programm sollte die Möglichkeit bieten, in einem Register am Anfang als Vorgabe die Anzahl der Ecken zu erhalten und dann daraus selbständig die Figur zu zeichen.

Welche Berechnungen sind dafür anzustellen? Wie landen wieder bei unserem bekannten Schildkrötensatz. Um mit einer gewissen Anzahl von Drehe-Befehlen, in unserem Fall jetzt der Anzahl der Ecken, in die Ausgangsrichtung zu blicken, ist der Vollkreis von 360 Grad zu durchlaufen. Auf jede der n Drehungen enfällt somit ein Winkel von 360 / n. Jetzt ist nur noch die Schrittweite festzulegen, der Einfachheit halber einigen wir uns hier auf einen festen Wert von 50 Schritten.

Die *Abb. 3.1.40* zeigt ein entsprechendes Programm, mit dem man ein beliebiges n-Eck zeichnen kann. Der Programmteil „POLY" velangt dazu im Register D3 die Anzahl der Ecken. Dabei muß ein Zahlenwert geladen sein, der in 360 aufgeht. Das Hauptprogramm „START" lädt dazu in diesem Beispiel den Wert 5. Die *Abb. 3.1.41* zeigt das Ergebnis dieses Programmes.

Sehen wir uns zum besseren Verständnis die einzelnen Befehle des Teiles „POLY" etwas genauer an. Zunächst legen wir die Konstante für unseren Schildkrötensatz, also den Wert 360, in dem Register D4 ab. Dann teilen wir diesen Wert durch die vorgegebene Anzahl der Ecken und erhalten somit den Winkelwert für die Drehe-Befehle. Um jetzt noch festzulegen, wieoft die Schleife „POLYSCHLEIFE" zu wiederholen ist, ziehen wir von der Eck-Anzahl eine Eins ab. Innerhalb der Schleife werden dann nur noch der Schreite-Befehl um den festgelegten Wert 50, sowie der Drehe-Befehl mit der errechneten Gradzahl, die im Register D4 übergeben wird, ausgeführt.

Interessantere Figuren entstehen mit dem Programm nach *Abb. 3.1.42.*

Hier wird durch die Anweisung ADD D0,D0, der Winkel verdoppelt und für eine weitere Drehe-Anweisung verwendet. Die *Abb. 3.1.43* zeigt das Ergebnis. Die Anzahl der Schleifendurchläufe ist hier so groß gewählt (200), daß sich geschlossene Figuren ergeben. Daher wird die Figur aber auch mehrfach durchlaufen. Wenn man das exakt haben will, so muß man die Durchlaufzahl berechnen. Hier sei die Anregung gegeben, einmal selbst die Berechnung zu versuchen.

Hinweis: Man beachte auch hier den Schildkrötensatz. Dabei kann die Schildkröte auch ein ganzzahliges Vielfaches von 360° durchlaufen, um wieder in die Ausgangslage zu kommen.

Wenn man jetzt zum Beispiel die Schrittzahl nach jedem Schleifendurchlauf verändert, so ergeben sich spiralförmige Gebilde. Zur Anregung der Phantasie zeigt die *Abb. 3.1.44* ein Beispiel. Das Programm „POLYSPI" erzeugt die *Abb. 3.1.45.* Der Winkel bleibt bei jedem Durchlauf konstant. Setzt man aber andere Werte ein, so ergeben sich sehr unterschiedliche Figuren. Die *Abb. 3.1.46* zeigt den Programmlauf für 125°. Versuchen Sie hier ruhig auch andere Variationen. Der Kreativität sind keine Grenzen gesetzt.

```
polyspi:
 move #2,d3      * Startwert
 move #200,d4    * Gesamtzahl
schleife:
 move d3,d0
 jsr @schreite
 move #144,d0    * Winkel
 jsr @drehe
 add #2,d3       * Laenge veraendern
 dbra d4,schleife
 rts
```

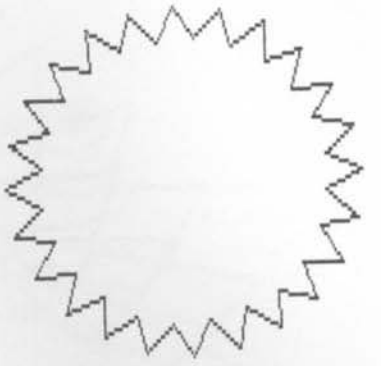


Abb. 3.1.43 Ergebnis mit Startwinkel = 125°

```
Textstart=009000 Fenster=009000 Tor=009000      amer CTRL-J=Hilfe
```

Abb. 3.1.44 Das Programm „polyspi"

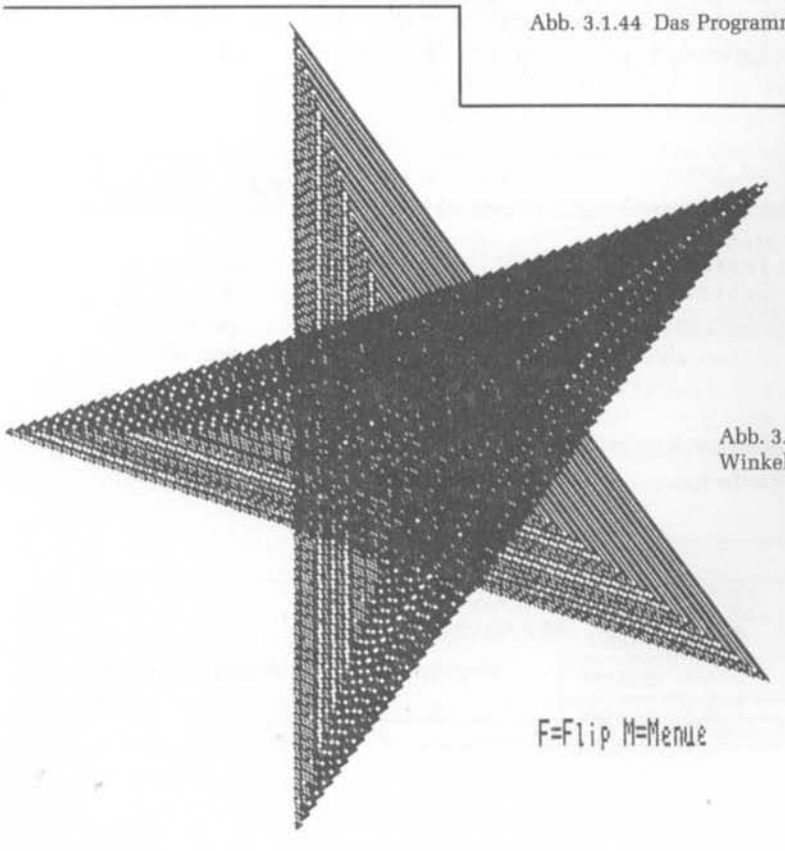
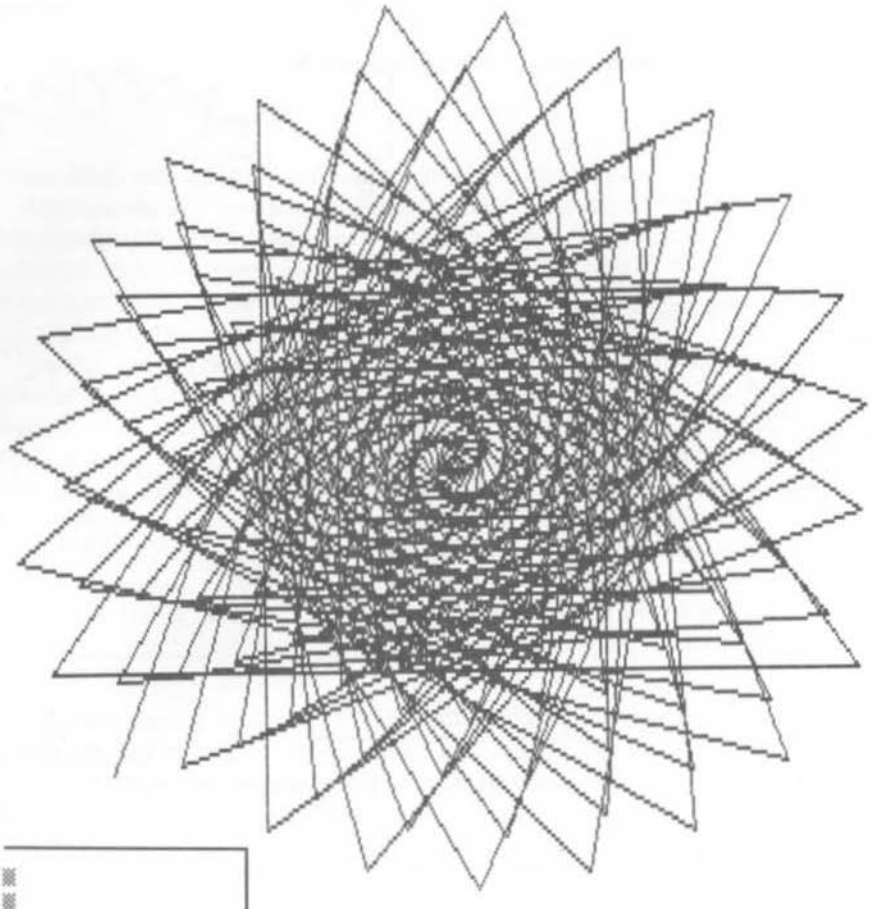


Abb. 3.1.45 Ergebnis bei einem Winkel von 144°



F=Flip M=Menue

Oben:
Abb. 3.1.46
Ergebnis bei
einem Winkel
von 125°

```
inspi:
 move #0,d3     * Winkel
 move #7,d4     * Increment
 move #400,d5   * Schleifendurchlaeufe
schleife:
 move #10,d0
 jsr @schreite
 move d3,d0
 jsr @drehe
 add d4,d3      * neuen Winkel bestimmen
 dbra d5,schleife
 rts
```

Links: Abb. 3.1.47
Das Programm „inspi"

Textstart=009000 Fenster=009000 Tor=009000 einf ämer CTRL-J=Hilfe

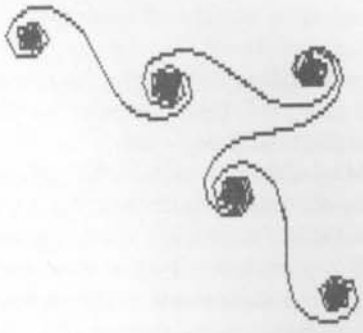


Abb. 3.1.48 Winkel = 0, Increment = 7

F=Flip M=Menue

Schildkrötengraphik

1	SCHREITE	D0.W=Anzahl
2	DREHE	D0.W=Winkel
3	HEBE	
4	SENKE	
5	SET	D1.W=X, D2.W=Y, D3.W=Winkel
19	SCHR16TEL	D0.W=Anzahl
47	HIDE	
48	SHOW	

Abb. 3.1.49 Zusammenfassung der bisherigen Schildkröten-Befehle

Angabe eines Wertes:

100	dezimal
$AFFF	sedezimal
%10100101	binär
100*3+5-4	Ausdruck
name	symbolisch
name+$AF	gemischt

Abb. 3.1.50 Beispiele für eine Wert-Angabe

Aufgabe:

Versuchen Sie, eine Systematik in die Figuren zu bringen. Verwenden Sie dazu Winkel von 45, 90, 180 und 270°. Dann nehmen Sie ähnliche Winkel, wie 44, 46, 89, 91 und 181° etc. Stellen Sie eine Tabelle auf.

Eine andere Möglichkeit ist es, den Winkel bei jedem Durchlauf zu verändern. Dann gelangt man zum Programm nach *Abb. 3.1.47*. Dabei wird ein Startwinkel angegeben und im Register D4 die Konstante, die bei jedem Durchlauf addiert wird. Die Schrittweite bleibt fest. In der *Abb. 3.1.48* ist das Ergebnis zu sehen.

Nun kann man die bekannten Elemente kombinieren, und es entstehen viele weitere Variationsmöglichkeiten. Es ist ratsam, sich mit dieser Thematik noch eine Weile zu beschäftigen und so die Kenntnis über die Befehle des Grundprogrammes sowie über die verwendeten Assemblerbefehle weiter zu vertiefen.

Die Grundprogrammbefehle „HIDE" und „SHOW"

Um die so entstehenden anspruchsvollen Programme noch weiter zu verfeinern, sei auf eine weitere Möglichkeit in unserer Zeichensprache hingewiesen. Mit dem Befehl „JSR @HIDE" ist es möglich, die Schildkrötendarstellung auszublenden. Der Befehl wirkt so, als ob man am Schluß des Programmlaufs „F=Flip" eingibt, wird aber schon während des Programmlaufs wirksam. Diese Unterdrückung der Schildkröte läßt sich durch den Aufruf von „JSR @HIDE" wieder rückgängig machen.

Am Schluß des Abschnitts 3.1 sei nochmals in der *Abb. 3.1.49* eine kurze Zusammenfassung der Zeichensprache-Befehle gegeben. Die Zahlen am Anfang geben den Index wieder, den man beim TRAP-Aufruf verwenden muß. Dieser Index wird aber vom Grundprogramm automatisch eingesetzt, wenn man das Zeichen „!" verwendet (siehe vorheriges Kapitel).

Die *Abb. 3.1.50* zeigt ein paar Beispiele für Wertangaben, wie sie mit dem Grundprogramm möglich sind und z. B. zum Laden eines Datenregisters (oder Adreßregisters etc.) verwendet werden können.

3.2 Schrittmotor steuern

In diesem Abschnitt wird beschrieben, wie man mit Hilfe eines Programms einen Schrittmotor zum Laufen bringt. Dabei sollte der Abschnitt auch dann durchgearbeitet werden, wenn man den Versuch selbst nicht ausführt, denn es gibt zahlreiche Programmbeispiele, die ohne zusätzliche Hardware laufen, und die für das Verständnis der folgenden Abschnitte notwendig sind.

Schrittmotoren sind wichtige Elemente von industriellen Steuerungen und Maschinen. Sie werden zum Beispiel verwendet, um Roboter anzutreiben, oder für elektronische Zeichenmaschinen.

Ein Schrittmotor ist zunächst ein Motor. Er unterscheidet sich von normalen Gleich- oder Wechselstrommotoren dadurch, daß er eine Anzahl fester Stellungen einnehmen

kann. Er kann sich daher nur schrittweise bewegen. Der daraus entstehende Vorteil liegt darin, daß es dadurch für den Computer möglich ist, die Motorachse in eine genau festgelegte Stellung zu bringen. Ebenso können natürlich auch exakte Umdrehungen der Achse vorgenommen werden.

Jetzt zum Aufbau eines Schrittmotors. Er besitzt einen Festmagneten auf seiner Achse. Mehrere Elektromagnete sind um ihn herum angeordnet. Die *Abb. 3.2.1* zeigt ein entsprechendes Schema, außerdem sind hier noch zwei Schalter und eine Spannungsquelle eingezeichnet. Mit den beiden Schaltern S1 und S2, die hier anstelle des Computers angeschlossen sind, kann man den Schrittmotor steuern. Die Stellung der Schalter bestimmt, welcher Strang gerade unter Strom steht. In der Abb. 3.2.1 sind der Strang P und R eingeschaltet. Durch die Stromrichtung wird ein magnetischer Nord- und Südpol am Stator erzeugt. Der Festmagnet an der Achse richtet sich entsprechend aus. Wenn man z. B. S1 umschaltet, so ergibt sich eine neue Orientierung, siehe *Abb. 3.2.2*. Der Anker hat sich um 90° gedreht, denn die Polarität am Stator PQ, also die Nord-Südrichtung, hat sich gedreht. Der Strom läuft nun in der Gegenrichtung, da die Spule Q anstelle von P durchflossen wird. Wenn man nun anschließend S2 umschaltet, so dreht sich der Motor wieder um 90° weiter, hier entgegen dem Uhrzeigersinn. Will man ihn nochmals drehen, so muß man S1 wieder umschalten. Dann S2 usw.

Hier sind immer zwei Spulen gleichzeitig unter Strom. Wenn man eine feinere Stufung erhalten will, so kann man den Stromkreis durch je einen weiteren Schalter auch noch wahlweise auftrennen. Dann ist zwischendurch auch nur mal eine Spule unter Strom, und man kann den Anker in 45° Schritten bewegen.

Aufgaben:

1. Wie sieht eine Schaltung aus, mit der man den Rotor in 45° Schritten bewegen kann?
2. Wie muß man die Schalter nacheinander stellen, um den Motor in 45° Schritten zu bewegen?
3. Wie kann man die Statoren konstruieren, um eine feinere Stufung zu erhalten, ohne die Anzahl der Spulen zu erhöhen? Hinweis: Man muß die Statoren aufteilen.

Neben dem Schrittmotor aus den Abb. 3.2.1 und 3.2.2, der als Unipolarschrittmotor bezeichnet wird, gibt es auch noch eine andere Art von Schrittmotoren, die bipolaren Schrittmotoren.

Die *Abb. 3.2.3* zeigt die Schaltung für die erste und die *Abb. 3.2.4* für die zweite Stellung. Zunächst fällt auf, daß gegenüber dem Unipolarmotor hier die Mittelanzapfung der Spule fehlt. Die Schalter sind nun so verdrahtet, daß man den Stromfluß durch die Spulen umpolen kann. Damit erreicht man den gleichen Effekt wie beim Unipolarmotor, nämlich, daß man die Nord-/Südrichtung vertauschen kann. Ein Unterschied besteht aber darin, daß man dazu nun die gleiche Spule verwendet; jetzt sind alle Wicklungen des Motors unter Strom. Damit ergibt sich ein besserer Wirkungsgrad, und bipolare Motoren sind für höhere Drehmomente ausgelegt, als entsprechende unipolare Motoren. Wichtig! Unipolare Motoren könnte man zwar auch so verschalten, daß sie

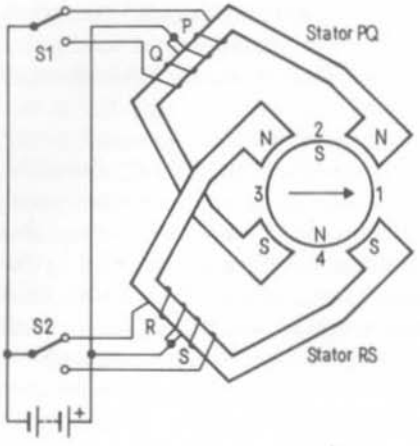


Abb. 3.2.1 P und R stehen unter Strom (unipolarer Schrittmotor)

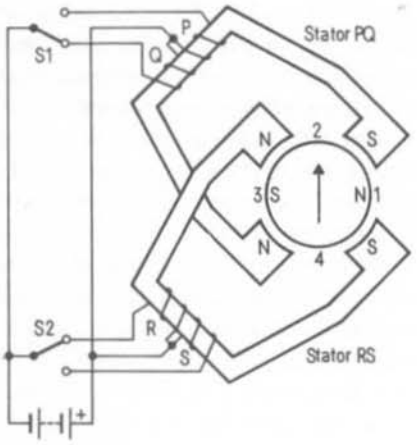


Abb. 3.2.2 Q und R stehen unter Strom (unipolarer Schrittmotor)

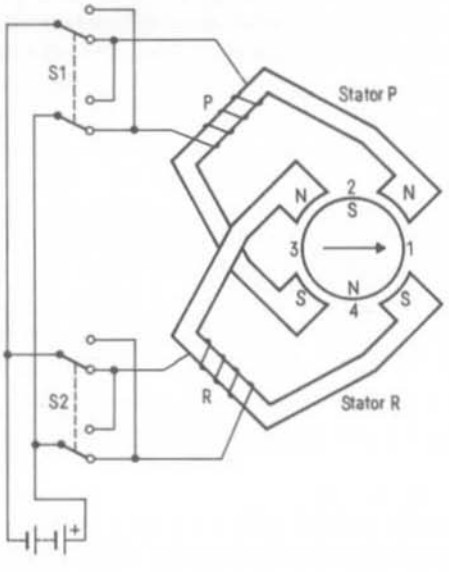


Abb. 3.2.3 Ein bipolarer Schrittmotor

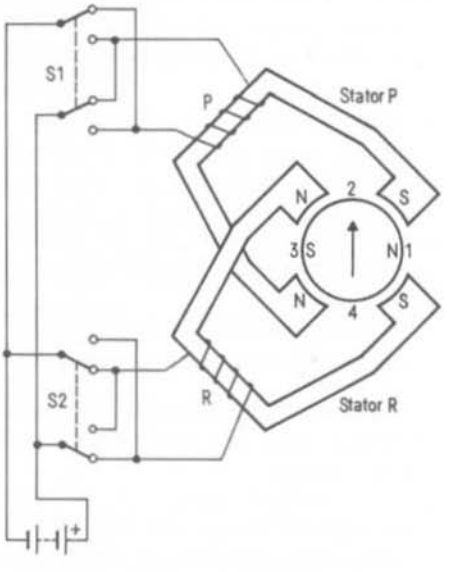


Abb. 3.2.4 P wurde gegenüber Abb. 3.2.3 umgepolt

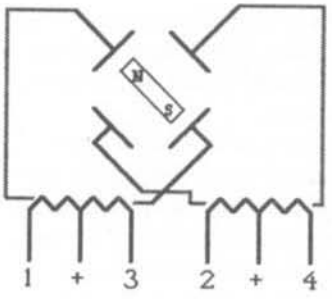


Abb. 3.2.5 Sechs Leitungen sind zum Betrieb nötig

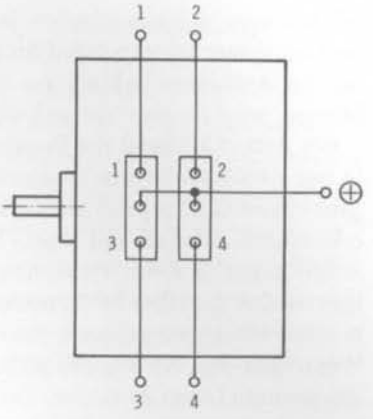


Abb. 3.2.6 Anschlußbeispiel eines Unipolar-Schrittmotors

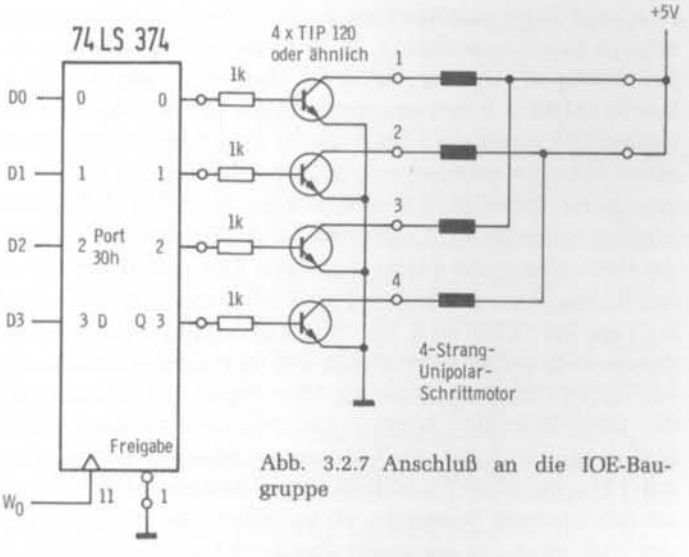


Abb. 3.2.7 Anschluß an die IOE-Baugruppe

b

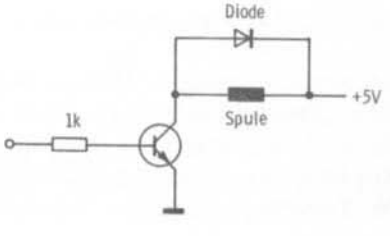


Abb. 3.2.8 Treiber mit „Freilauf"-Diode

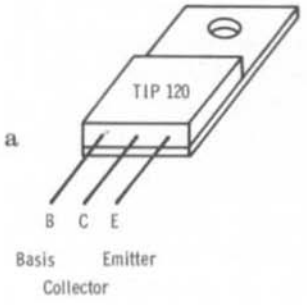
wie bipolare Motoren arbeiten, jedoch erzeugen die Ströme in den Wicklungen Wärme; und unipolare Motoren sind nicht dafür ausgelegt, alle Spulen unter Strom zu halten. Da die Ansteuerschaltung für bipolare Motoren komplizierter ist als für unipolare Motore, wird für den Versuch ein unipolarer Motor verwendet.

Die *Abb. 3.2.5* zeigt die Draufsicht eines Schrittmotors mit einem Anschlußschema; in der *Abb.3.2.6* ist die Innenschaltung dazu zu sehen. Leider ist die Belegung nicht ganz einheitlich; so muß man sich entweder beim Hersteller nach der genauen Belegung erkundigen, oder sie mit einem Vielfachinstrument ausmessen. Es gibt beim 4-Strang-Schrittmotor ja zwei Wicklungen. Diese beiden Wicklungen sind gegeneinander isoliert, so daß man dies leicht messen kann. Dann bleiben je drei Anschlüsse übrig, die zu je einer Wicklung gehören. Einer der Anschlüsse ist die Mittelanzapfung der Spule. Wenn man den Widerstand mißt, so ist er vom Mittelpunkt aus halb so groß, wie über die gesamte Länge gemessen. Die Orientierung kann man zwar nicht messen, aber man kann sie ausprobieren. Wenn man z.B. 1 und 3 vertauscht, so ändert sich nur die Drehrichtung des Schrittmotors.

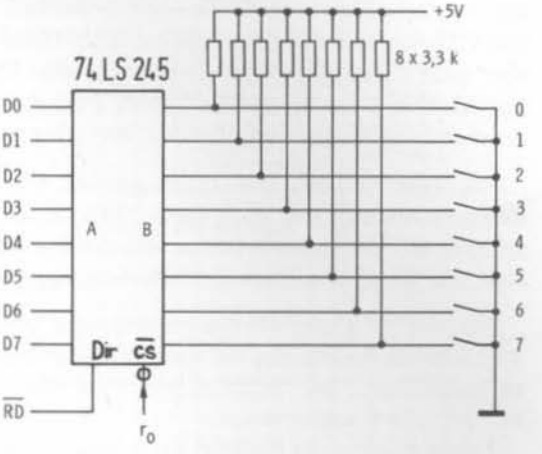
Den Schrittmotor kann man nicht direkt an den Computer anschließen. Man braucht dazu ein sogenanntes Interface, also eine Schaltung, die den Anschluß ermöglicht. Dabei werden vier Ausgangsleitungen vom Computer benötigt. Dazu kann man die IOE-Baugruppe (siehe Kapitel 7) verwenden. Sie besitzt 16 Ausgänge und 16 Eingänge. Die Ausgänge kann man aber auch nicht direkt mit dem Schrittmotor verbinden, dazu müssen Leistungstreiber, z. B. in Form von Transistoren nachgeschaltet werden. Ein Schaltbeispiel zeigt die *Abb. 3.2.7*. Hier ist nur ein Port der IOE-Baugruppe abgebildet. Das IC 74LS374 liefert an seinen Ausgängen die Daten an. Man benötigt vier Bits, also werden die Ausgänge 0 bis 3 an die Transistor-Treiber geschaltet. Dies geschieht über einen Schutzwiderstand von je 1kΩ. Als Transistoren werden hier die Typen TIP120 verwendet. Dabei handelt es sich um sogenannte Darlington-Leistungstransistoren. Im Inneren verbergen sich eigentlich zwei Transistoren. Diese Transistoren TIP120 könnten auch sehr große Lasten vertragen und sind daher für solche Versuche besonders empfehlenswert. Auch den Typ TIP110 kann man verwenden. Der TIP 120 kann 2 Ampere bei 100V und der TIP 110 2 Ampere bei 60 Volt schalten. Für unsere Anwendung genügt etwa 1/1000 des möglichen Ausgangsstroms. Bei den Transistoren handelt es sich um sogenannte NPN-Typen, die also mit einer positiven Spannung an der Basis geschaltet werden. Die Spulen sind vom Kollektor-Ausgang nach +5 V geschaltet. Wenn man die Leistung der Motoren ausnützen will, so kann man sie auch mit +12V betreiben, je nach Motortyp. Um die Funktion zu testen, ist es jedoch besser, sie mit niederer Spannung zu betreiben, da zunächst nicht garantiert ist, daß nur maximal zwei Spulen unter Strom stehen. Das ist erst nach Inbetriebnahme der Software der Fall.

Der Pin 1 des ICs 74LS374 muß mit 0V verbunden werden, sonst ergeben sich an den Ausgängen keine Signale.

Manchmal sind Schaltungen zu sehen, in denen über die Spulen der Motoren Dioden geschaltet sind (*Abb. 3.2.8*). Dies sind sogenannte „Freilauf-Dioden"; sie haben eine Schutzfunktion. Wenn der Transistor stromlos wird, also sperrt, so induziert das zusammenbrechende Magnetfeld in der Spule eine hohe Spannung, die genau entgegen der ursprünglichen Spannung liegt. Wenn der Transistor nur kleine Spannungen

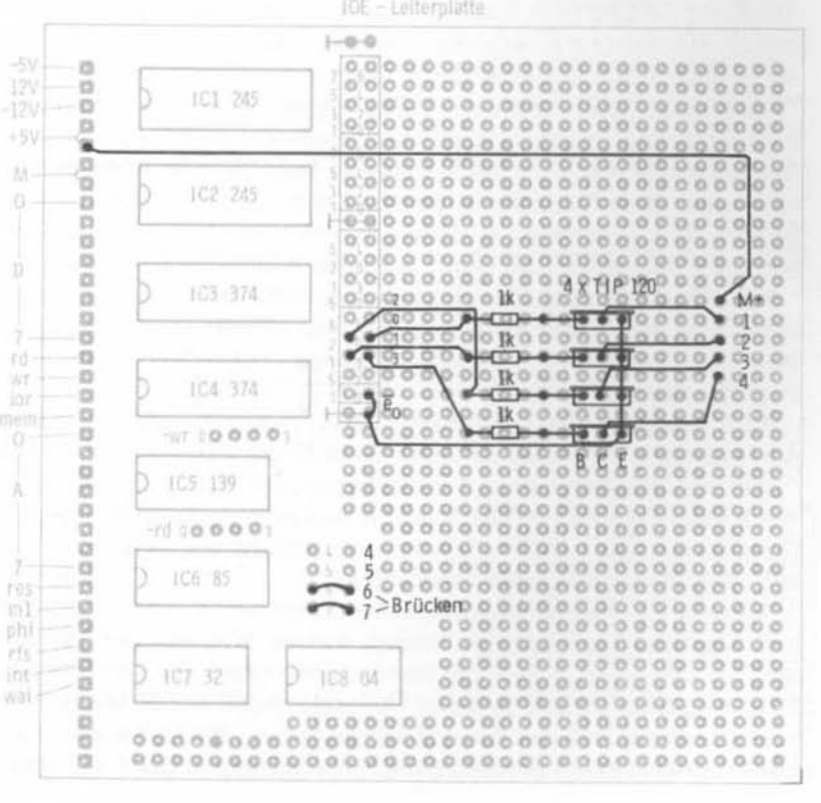


Oben: Abb. 3.2.9 Anschlußschema des TIP120 (TIP110)



Oben: Abb. 3.2.11 Anschluß von 8 Tastern an die IOE-Baugruppe

Unten: Abb. 3.2.10 Anordnung und Verdrahtung der Treiber auf der IOE-Baugruppe



verträgt, so würde er dadurch zerstört werden. Wir verwenden aber den TIP 110, mit 100 V Sperrspannung, dem man auf diese Weise nicht so leicht was anhaben kann. Wer aber ganz sicher gehen will, schaltet je eine Diode antiparallel, also parallel, aber in Sperrrichtung, zur Spule. Die Katode muß also an der Versorgungsspannung, also am Plusanschluß liegen, denn für den normalen Betrieb muß die Diode ja gesperrt sein.

Nun zum Aufbau. Zum einen gibt es eine fertige Leiterplatte, die auch für die Robotersteuerung aus dem Buch „Mikrocomputer selbstgebaut und programmiert" geeignet ist. Auf dieser befinden sich die Transistoren fertig verschaltet. Einfacher ist es aber, die Verdrahtung auf der IOE-Baugruppe vorzunehmen.

Die *Abb. 3.2.9* zeigt die Pin-Belegung des Transistors TIP 120 (TIP 110). Aus der *Abb. 3.2.10* ist die Anordnung auf der IOE-Baugruppe mit allen notwendigen Verdrahtungen zu entnehmen. Der Schrittmotor wird über weitere Leitungen an die Anschlüsse M,1,2,3 und 4 angeschlossen.

Achtung, nicht die Brücken bei 6 und 7 vergessen, sonst wird später die Baugruppe nicht richtig adressiert. Ebenfalls wichtig ist die kleine Brücke bei e0-Quer. Sie sorgt dafür, daß an den Ausgängen des Ports 74LS374 auch TTL-Pegel liegen. Wenn man die Brücke vergißt, so sind die Ausgänge offen (TRI-State).

Für die weiteren Versuche werden wir nicht die Tastatur, sondern ein kleines Tastenfeld verwenden. Dieses Tastenfeld soll ebenfalls auf der gleichen IOE-Baugruppe aufgebaut werden. Dazu benötigt man 8 einzelne Drucktaster, z.B. vom Typ Mini-Digitast o.ä. *Abb. 3.2.11* zeigt die Schaltung. Als Port wird das IC 74LS245 verwendet, das auf der IOE-Baugruppe schon entsprechend beschaltet ist. Die eine Seite der Tastenkontakte wird jeweils mit einem Eingang des Ports verbunden, die zweite Seite liegt gemeinsam an Masse. Acht Widerstände sorgen dafür, daß bei offenem Kontakt eine Spannung von +5 V am Porteingang liegt. Die Schaltung würde normalerweise auch ohne diese Widerstände funktionieren, da offene TTL-Eingänge immer eine logische Eins annehmen, in diesem Zustand sind sie aber besonders empfindlich gegenüber Störungen. Also besser ist es, die Widerstände mit einzubauen.

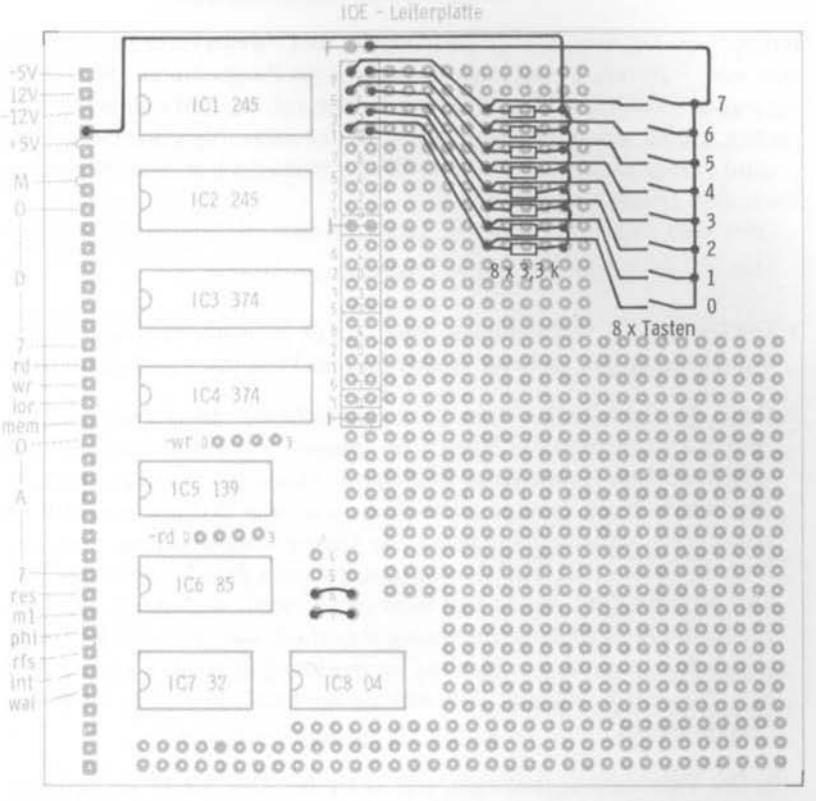
Die *Abb. 3.2.12* zeigt das Verdrahtungsschema auf der IOE-Baugruppe. Die Tasten selbst kann man besser auf einer getrennten Leiterplatte unterbringen, die man über eine lange Leitung mit der IOE-Baugruppe verbindet. Die *Abb. 3.2.13* zeigt eine mögliche Anordnung.

Auch wenn man den Schrittmotor-Versuch nicht durchführt, sollte man zumindest das zusätzliche Tastenfeld aufbauen, da es für weitere Versuche benötigt wird.

Nun zur Aufgabenstellung. Durch Drücken der Taste 0 soll sich der Motor drehen. Wenn man die Taste 1 drückt, so soll er sich in die entgegengesetzte Richtung drehen. Obwohl wir nicht wissen, wie herum vorwärts ist, soll Taste 0 die Vorwärtsrichtung sein, also Taste 1 die Rückwärtsrichtung.

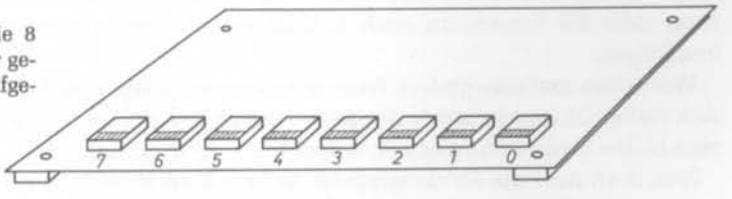
Wie geht man bei der Programmentwicklung vor? Zunächst muß die Aufgabenstellung klar sein. Dann folgt die Problemanalyse. Wie man den Motor bedienen muß, damit er sich dreht, ist klar; die Tasteneingabe haben wir auch schon definiert.

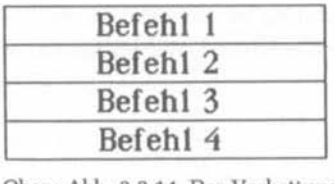
Dann werden wir der besseren Übersicht wegen ein Struktogramm erstellen, das zunächst einmal grob den Programmablauf festlegt. Nachher wird der Ablauf verfeinert und schließlich in ein Assemblerprogramm übertragen. Das Assemblerprogramm wird



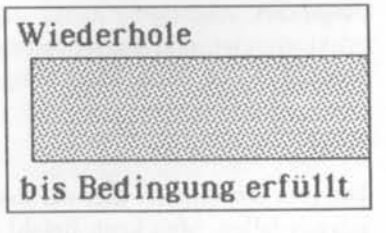
Oben: Abb. 3.2.12 Anordnung und Verdrahtung der 8 Tasten auf der IOE-Baugruppe

Rechts: Abb. 3.2.13 Die 8 Tasten können auf einer getrennten Leiterplatte aufgebaut werden





Oben: Abb. 3.2.14 Der Verkettungs-Baustein



Rechts: Abb. 3.2.15 Der Schleifen-Baustein

eingegeben, mit dem Assembler übersetzt und - wenn keine syntaktischen Fehler mehr vorliegen – gestartet. Nun folgt der eigentliche Programmtest. Man sollte alle Kombinationsmöglichkeiten prüfen. Tritt ein Fehler auf, so muß man zunächst das Syntaxdiagramm prüfen, dann die Übertragung in das Assemblerprogramm.

Sind Fehler vorhanden, so sind diese systematisch zu suchen, und es ist dann einer nach dem anderen zu beseitigen.

Doch nun zum Struktogramm. Dazu ein paar allgemeine Dinge vorab.

Man unterscheidet dabei drei Programmstrukturen.

1. Die Verkettung von Befehlen. *Abb. 3.2.14* zeigt das Struktogramm. Dabei wird ein Befehl hinter den anderen gesetzt und im Programm nacheinander ausgeführt.

2. Dann der Schleifenbaustein. Man unterscheidet dabei nochmals zwei Arten. Nach *Abb. 3.2.15* werden die Befehle im inneren Block durchlaufen, bis eine Bedingung erfüllt ist. Man nennt diese Bedingung Abbruchbedingung. Nach *Abb. 3.2.16* wird der innere Block solange wiederholt, wie eine Bedingung erfüllt ist. Dies ist eine sogenannte Schleifenbedingung. Der Unterschied ist gering, aber von großer Bedeutung. Im ersten Fall wird die Abfrage erst nach dem Durchlaufen des inneren Blocks durchgeführt. Im zweiten Fall wird zunächst die Bedingung abgefragt und dann der Block durchlaufen, falls die Bedingung erfüllt war. Im ersten Fall wird der Block also mindestens einmal durchlaufen, im zweiten Fall wird er nicht unbedingt durchlaufen. War die Bedingung von Anfang an nicht erfüllt, so wird der Block einfach übersprungen.

3. Der Verzweigungsbaustein, wie er in der *Abb. 3.2.17* als Beispiel dargestellt ist. Hier wird in Abhängigkeit von einer Bedingung der Ja-Teil oder der Nein-Teil durchlaufen. Damit lassen sich also Entscheidungen realisieren.

Mit diesen drei Strukturen kann man alle Programm-Aufgaben bewältigen. Dabei kann man die Strukturen auch ineinander schachteln, um komplexe Aufgaben zu bewältigen.

Wer schon mal eine andere Programmiersprache kennengelernt hat, z. B. Basic, wird sich vielleicht wundern, wie das geht, ohne GOTO, doch es ist möglich, wenn man auch eine kleine Umgewöhnungszeit braucht.

Nun muß man die Struktogramme natürlich auch noch in ein Assemblerprogramm umwandeln. Dazu braucht man natürlich Sprünge (also GOTO), aber dadurch, daß man sich an ein Struktogramm hält, werden die Sprünge nur in wohlgeordneter Form ausgeführt. Wichtig ist dabei, daß die grundlegende Ordnung in unserem Programm strukturiert ist, sich also an den Regeln des Struktogrammes orientiert. Wildes Hin- und Herspringen ist daher nicht nötig und sehr verpönt.

1. Die Verkettung

Für den ersten Fall, den wir kennengerlernt haben, wird uns die Übersetzung nicht schwer fallen. Man kann Befehl für Befehl aus dem Struktogramm in den Assemblercode umsetzen. Sprünge werden nicht benötigt, ggf. nur Unterprogrammaufrufe.

Abb. 3.2.16 Ein weiterer Schleifen-Baustein

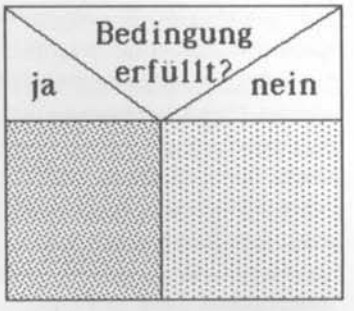


Abb. 3.2.17 Der Verzweigungs-Baustein

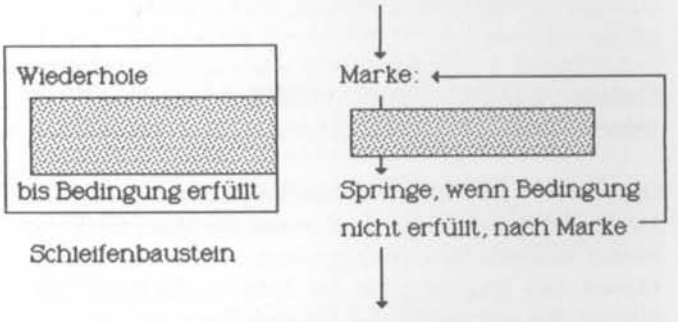


Abb. 3.2.18 Umsetzen des Schleifen-Bausteins

```
* Der Schleifen-Baustein

marke:          * Wiederhole
 move #1,d0     *    Befehl 1
 jsr @schreite  *    Befehl 2
 move #1,d0     *    Befehl 3
 jsr @drehe     *    Befehl 4
 jsr @csts      * ( Bedingung erzeugen )
 beq marke      * bis Taste gedrueckt
 rts            * Ende
```

Abb. 3.2.19 Assemblerbeispiel für einen Schleifen-Baustein

```
Textstart=009000 Fenster=009000 Tor=009000      amer CTRL-J=Hilfe
```

2. Der Schleifenbaustein

Beim Schleifenbaustein sieht das anders aus. Die *Abb. 3.2.18* zeigt das Umsetzungsschema. Zunächst beginnt man am Anfang des Blockes mit der Definition einer Marke. Diese Marke kann natürlich einen beliebigen Namen haben. Wir erinnern uns, mit einer Marke kennzeichnen wir eine Stelle im Programm, also eine bestimmte Adresse innerhalb unseres nach der Übersetzung durch den Assembler entstehenden Maschinencodes, durch einen symbolischen Namen. Dann wird der innere Block der Schleife übertragen. Dort können natürlich auch weitere Strukturen außer der Verkettung vorkommen, die man systematisch übertragen muß. Am Schluß erfolgt die Abfrage der Bedingung, die man gegebenenfalls erst bereitstellen muß, und dann folgt ein Sprung zurück zur Marke, die am Anfang des Strukturblocks definiert wurde, wenn die Bedingung nicht erfüllt ist. Ist sie erfüllt, so soll die Schleife ja nicht erneut wiederholt werden, sondern es erfolgt der Abbruch.

Der Grundprogrammbefehl CSTS
Der Assemblerbefehl BEQ

Die *Abb. 3.2.19* zeigt ein Assemblerprogramm, das nach diesem Schema konstruiert wurde. Bei dieser Gelegenheit lernen wir einen neuen Grundprogrammbefehl, also ein bereits vorliegendes Unterprogramm, sowie einen neuen Assemblerbefehl des 68000/8 kennen. Das Programm hat die Aufgabe, die Schildkröte entlang einer Kreisbahn zu führen. Das soll geschehen, bis eine Taste auf der Tastatur gedrückt wird. Mit dem Unterprogramm „CSTS", kann man die Bedingung „Taste gedrückt" abfragen. Das Unterprogramm liefert den Wert $FF im Register D0.B ab, wenn eine Taste gedrückt wurde, sonst den Wert 0. Gleichzeitig werden die sogenannten Bedingungsflags in dem Status-Register des 68000/8 gesetzt. Im 68000/8 gibt es mehrere solcher Bedingungsflags. Einmal das Z-Flag, oder Zero, Null-Flag. Es wird gesetzt, wenn der Wert 0 als Ergebnis auftaucht. Dann das N-Flag, oder Negativ-Flag. Es wird gesetzt, wenn das Vorzeichenbit eines Ergebnisses gesetzt ist. Dabei geht man von einer Zahlendarstellung im Zweierkomplement aus. Dann das C-Flag, oder Carry-Flag; es wird gesetzt, wenn ein Übertrag entsteht, z.B. wenn man die Zahl 5 von der Zahl 3 subtrahiert. Dabei wird die Zahl als Zahl ohne Vorzeichen betrachtet. Dann gibt es noch das V-Flag, oder Overflow-Flag. Es wird gesetzt, wenn ein arithmetischer Überlauf entsteht; die Zahl wird dabei als Zweierkomplement-Zahl aufgefaßt. Man darf es nicht mit dem C-Flag verwechseln.

Der richtige Umgang mit den Bedingungsflags ist nicht ganz einfach. Man lernt das nur durch viel Übung nach und nach.

Man kann diese Bedingungsflags mit den bedingten Sprüngen direkt abfragen. Unter „bedingter Sprung" versteht man einen Befehl, der nur dann einen Sprung zum angegebenen Ziel ausführt, wenn eine Bedingung erfüllt ist. In unserem Fall ist das der Befehl „BEQ". „BEQ" ist die Abkürzung für „branch equal zero" oder übersetzt: „springe, wenn Null". Der Sprung zur angegebenen Marke erfolgt nur, wenn das Null-Flag gesetzt war, und das geschieht genau dann, wenn das Unterprogramm „CSTS" den

Wert 0 im Register D0.B abgelegt hat, also keine Taste gedrückt war. Wenn aber der Wert FF vorliegt, weil eine Taste gedrückt war, so ist die Bedinung nicht erfüllt, der BEQ-Befehl veranlaßt keinen Sprung, und die Schleife wird beendet.

Das Programm zeichnet einen Kreis. Dabei wird das Programm erst dann beendet, und es meldet sich auch erst dann mit „F=Flip M=Menue", wenn man irgend eine Taste der Tastatur gedrückt hat.

Die *Abb. 3.2.20* zeigt die Umsetzung der zweiten Schleifenart. Das ist also die Schleife, bei der abgebrochen werden soll, wenn eine Bedingung erfüllt ist. Hier benötigt man zwei Marken. Einmal eine Marke, die als Einsprung für die Wiederholung der Schleife dient, und dann eine Marke, die zum Verlassen der Schleife benötigt wird. Die *Abb. 3.2.21* zeigt ein entsprechendes Programmbeispiel.

Nicht immer ist es ganz klar, welchen der beiden Schleifentypen man verwenden soll. So auch in diesem Beispiel. Denn das Programm verhält sich fast genauso, wie im ersten Beispiel. Die Aufgabe ist eigentlich die gleiche: Die Schildkröte soll entlang einer Kreisbahn laufen, bis eine Taste gedrückt wird. Aber genauer müßte man hier formulieren: Wenn keine Taste gedrückt ist, dann soll die Schildkröte entlang einer Kreisbahn laufen. Der Unterschied besteht darin, daß im ersten Fall mindestens ein Schritt und eine Drehung durchgeführt wird, auch wenn die Taste schon von Anfang an gedrückt war. Bei dem neuen Beispiel aber wird das Programmstück einfach übersprungen, wenn die Bedingung sofort erfüllt ist.

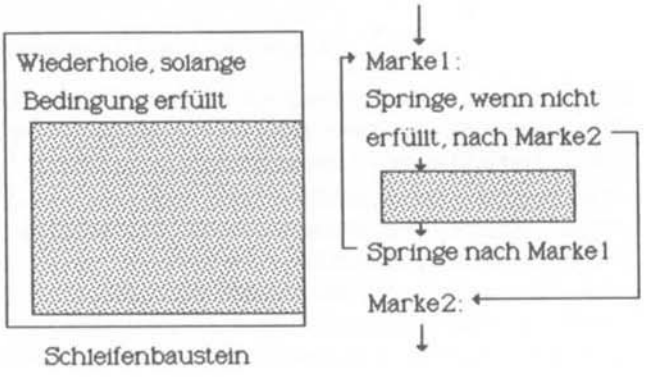
3. Der Verzweigungsbaustein

Die Umsetzung des Verzweigungsbausteins ist nicht ganz einfach. Die *Abb. 3.2.22* zeigt das Schema. Im Struktogramm sind zwei Programmteile, der Ja-Teil und der Nein-Teil, nebeneinander geschrieben. Im Assemblerprogramm müssen diese beiden Teile aber nacheinander geschrieben werden. Man beginnt mit der Abfrage. Wenn die Bedingung nicht erfüllt ist, so springt man zur „Marke1", dort steht der Nein-Teil. Hinter der Abfrage steht dann der Ja-Teil, denn er wird ausgeführt, wenn die Bedingung erfüllt war. Hinter dem Ja-Teil steht ein weiterer Sprung, zur „Marke2", denn der Nein-Teil muß ja übersprungen werden. Hinter dem Nein-Teil beginnt nach der „Marke2" wieder der gemeinsame Bereich. Die Abb. 3.2.23 zeigt ein Programmbeispiel für den Verzweigungsbaustein.

Der Grundprogrammbefehl CI
Die Assemblerbefehle CMP und BNE

Ein Zeichen soll von der Tastatur gelesen werden. Wenn man das Zeichen „A" (groß A) eingegeben hat, so soll eine senkrechte Linie auf dem Bildschirm erscheinen, sonst eine waagrechte Linie. Zum Einlesen eines Zeichens wird ein neues Unterprogramm benötigt. Das Unterprogramm „CI" liest ein Zeichen in das Register D0.B. Dabei wartet das Unterprogramm solange, bis eine Taste auf der Tastatur gedrückt wurde.

Abb. 3.2.20
Der Schleifen-Baustein II

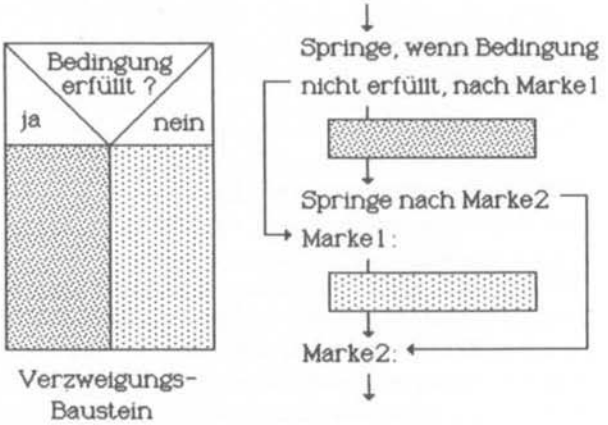



```
* Der Schleifen-Baustein

marke1:          * Wiederhole solange
 jsr @csts       * Taste nicht gedrueckt
 bne marke2
 move #1,d0      *    Befehl 1
 jsr @schreite   *    Befehl 2
 move #1,d0      *    Befehl 3
 jsr @drehe      *    Befehl 4
 bra marke1
marke2:
 rts             * Ende
```

Abb. 3.2.21 Beispiel
zum Schleifen-Baustein II

```
Textstart=009000 Fenster=009000 Tor=009000      amer CTRL-J=Hilfe
```

Abb. 3.2.22
Der Verzweigungs-Baustein



```
▒
▒

* Der Verzweigungs-Baustein

start:              * Bitte Zeichen eingeben
 jsr @ci            * Zeichen als Bedingung
 cmp.b #'A',d0      * Wenn Buchstabe A
 bne marke1         * Dann JA-Teil
 move #100,d0       * und Linie zeichnen
 jsr @schreite      * die senkrecht ist
 bra marke2         * NEIN-Teil ueberspringen
marke1:
 move #90,d0        * Sonst waagrechte Linie
 jsr @drehe         * zeichnen
 move #100,d0
 jsr @schreite      * im NEIN-Fall
marke2:
 rts                * Ende

▒
▒
▒
Textstart=009000 Fenster=009000 Tor=009000 einf amer CTRL-J=Hilfe
```

Abb. 3.2.23
Programmbeispiel zum Verzweigungs-Baustein

BEQ	equal, gleich	Z
BNE	not equal, ungleich	$\overline{Z}$
BGT	greater than, größer	$N*V*\overline{Z}+\overline{N}*\overline{V}*\overline{Z}$
BGE	greater or equal, größer gleich	$N*V+\overline{N}*\overline{V}$
BLE	less or equal, kleiner gleich	$Z+N*\overline{V}+\overline{N}*V$
BLT	less, kleiner	$N*\overline{V}+\overline{N}*V$
BHI	high, mehr	$\overline{C}*\overline{Z}$
BHS	high or same, mehr oder gleich	$\overline{C}$
BLS	less or same, weniger oder gleich	$C+Z$
BLO	less, weniger	C
BVC	overflow clear, kein Überlauf	$\overline{V}$
BVS	overflow set, Überlauf	V
BPL	plus, positiv	$\overline{N}$
BMI	minus, negativ	N
BCC	carry clear, kein Carry	$\overline{C}$
BCS	carry set, Carry	C

Abb. 3.2.24
Zusammenstellung möglicher Sprungbefehle

Mit dem neuen Assemblerbefehl „CMP.B", der die Abkürzung von „compare", also „vergleiche" ist, wird der Vergleich mit dem Zeichen „A" durchgeführt. Dabei bewirkt der Befehl „CMP" eine Subtraktion des Wertes der linken Seite von der rechten.
Es wird hier also der ASCII-Wert des Buchstabens „A" vom Inhalt des Registers D0.B subtrahiert. Im Register D0.B steht der ASCII-Wert des eingelesenen Zeichens. Das Subtraktionsergebnis wird beim „CMP„-Befehl nicht abgespeichert, sondern es werden nur die Bedingungsflags im Status-Register gesetzt. Wenn also das Zeichen „A" von der Tastatur eingegeben wurde (SHIFT A drücken), so steht im Register D0.B der ASCII-Wert des Zeichens „A". Dann wird durch den Befehl „CMP.B #'A',D0" ebenfalls der ASCII-Wert von „A" vom Inhalt des Registers D0.B subtrahiert, und es ergibt sich der Wert 0. Damit wird das Null-Flag gesetzt, und der Sprung „BNE" (branch non equal, also springe, wenn nicht Null) wird nicht ausgeführt. Das Programm läuft in den Ja-Teil.

Dort wird die senkrechte Linie gezeichnet und mit „BRA MARKE2" ein Sprung zum Programmende bei „MARKE2" durchgeführt.

Wenn nach dem Programmstart ein anderer Buchstabe eingetippt wird, z. B. ein kleines „a", so liefert die Abfrage die Bedingung „Nicht-Null", und der Sprung „BNE", also „springe, wenn nicht Null", wird ausgeführt. Das Programm gelangt bei der Ausführung dann zum Nein-Teil. Dort steht ein Drehe-Befehl, der die Schildkröte in die waagrechte Richtung dreht, und dann wird die Linie gezeichnet.

In der Praxis gibt es auch Verzweigungsbausteine, die nur einen Ja- oder einen Nein-Teil besitzen. Dann kann man die Umsetzung natürlich vereinfachen. Auch unser Beispielprogramm läßt sich so formulieren, daß man nur einen Nein-Teil braucht:

Wenn die Taste „A" gedrückt wird, dann soll keine Drehung um 90 Grad durchgeführt werden.

Die Linie wird dann in einer Verkettung angefügt, da sie immer gezeichnet wird.

Aufgabe:

1. Schreiben Sie ein Struktogramm, das der obigen Formulierung entspricht.
2. Setzen Sie das Struktogramm in ein Programm um. Das Programm muß sich genauso verhalten wie das aus *Abb 3.2.23*, obwohl es eine andere Struktur besitzt.

Der 68000/8 besitzt eine Reihe von unterschiedlichen Sprungbefehlen. Die *Abb. 3.2.24* zeigt eine Zusammenstellung möglicher Sprungbefehle, die aber erst nach und nach in den Programmen vorkommen und nicht alle auf einmal verstanden werden müssen.

Die ersten beiden Sprünge in der Abb. 3.2.24 sind uns schon bekannt. Damit kann man das Z-Flag prüfen, also ob der Wert 0 nach einer Operation vorliegt oder nicht. Die nächsten vier Befehle dienen der Abfrage bei arithmetischen Operationen mit Zweierkomplement-Zahlen. So kann man zwei Zahlen miteinander vergleichen, wenn man vor dem Sprung den Befehl „CMP" verwendet. Die nächsten vier Befehle dienen dem Vergleich von Zahlen, die nicht im Zweierkomplement vorliegen, also ohne Vorzeichen sind. Dann kann man die Flags aber auch noch einzeln prüfen. Der Befehl „BCC" ist übrigens identisch mit dem Befehl „BHS", und der Befehl „BCS" identisch mit dem

```
start:              * Wiederhole
 move #100,d0       *  Linie zeichnen
 jsr @schreite      *  vorwaerts
 move #-100,d0      *  und
 jsr @schreite      *  rueckwaerts
 jsr @csts          *  Bedingung einlesen
 beq keinetaste     *  Wenn Taste gedrueckt
 jsr @erapen        *  erst mal alte Position
 move #100,d0       *  loeschen
 jsr @schreite      *
 move #-100,d0      *
 jsr @schreite      *  dann Taste abfragen
 jsr @ci            *  dazu Wert einlesen
 cmp.b #'R',d0      *  Grosser Buchstabe
 bne warnichtr      *  Rechts
  move #-10,d0      *   also z.B. -10 laden
  bra wardochr      *
warnichtr:          *  Links annehmen
 move #10,d0        *   also z.B. +10 laden
wardochr:           *  weiter hier
 jsr @drehe         *  ausfuehren der Drehung
 jsr @setpen        *  wieder auf Schreiben
keinetaste:         *  sonst nichts weiter
 bra start          * Endlos
```

```
Textstart=009000 Fenster=009000 Tor=009000 einf amer CTRL-J=Hilfe
```

Abb. 3.2.25
Kombiniertes
Programmbeispiel

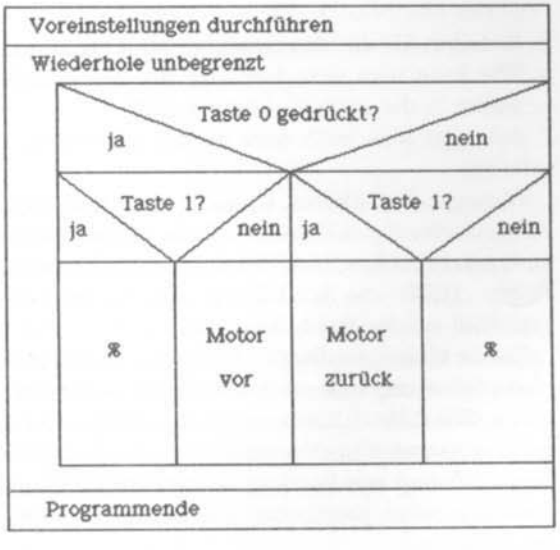


Abb. 3.2.26
Struktogramm –
Schrittmotor

Befehl „BLO". Die beiden unterschiedlichen Namen wurden eingeführt, um Assemblerprogramme übersichtlicher zu gestalten; so weiß man bei BLO, daß man einen arithmetischen Vergleich abfragen will, und bei BCS, daß man nur das Carry-Flag prüfen will, z.B. nach einer Schiebeoperation, die nichts mit Arithmetik zu tun hat. Nicht alle Befehle des 68000/8 verändern die Flags, und das ist auch gut so. Oft kann man eine Abfrage erst später durchführen und will alte Flags erhalten. Auf die Flags wirken die „MOVE„-Befehle und alle arithmetischen Operationen, wie „SUB", „ADD" oder „CMP".

Die *Abb. 3.2.25* zeigt noch ein kombiniertes Beispiel.

Die Grundprogrammbefehle SETPEN und ERAPEN

Diese beiden Befehle lernen wir in diesem Beispiel neu kennen. „ERAPEN" schaltet den Graphik-Prozessor auf „Löschen", alle Linien, die dann gezeichnet werden, werden danach schwarz geschrieben, das heißt, falls sich auf dem Bildschirm eine helle Linie befand, wird diese gelöscht. Nach „SETPEN" wird der Schreibstift wieder auf „Weiß" geschaltet.

Das Programm läßt einen Zeiger im Uhrzeigersinn oder entgegen dem Uhrzeigersinn drehen. Wenn man die Taste „R" drückt, so dreht sich der Zeiger im Uhrzeigersinn um je 10°, und wenn man eine andere Taste drückt, so dreht er sich umgekehrt.

Aufgabe:

1. Erstellen Sie das dazugehörige Struktogramm.
2. Wie kann man erreichen, daß der Zeiger sich auch nach dem Loslassen der Taste weiter in die neue Richtung dreht?

   Achtung: Man muß dazu einiges im Programm ändern, jedoch bleibt die Struktur erhalten.

Hinweis: Bei älteren Grundprogrammen (3.2) kann es vorkommen, daß bei der Ausgabe des Schildkrötensymbols automatisch wieder auf SETPEN geschaltet wird. Dies geschieht alle 20 Millisekunden. Will man das verhindern, so muß man mit dem Befehl „HIDE" die Schildkröte zunächst ausblenden und sie mit „SHOW" nach dem Löschteil wieder einblenden.

Dieser kleine Ausflug in ein grundsätzliches Themß war notwendig, um die komplexe Steuerung unseres Schrittmotors verstehen zu können. Nun zurück zum Schrittmotor. Die *Abb. 3.2.26* zeigt das Struktogramm der Schrittmotorsteuerung. Es beginnt mit der Anweisung „Voreinstellungen durchführen". Dort müssen zum Beispiel Variable initialisiert werden. Solche Voreinstellungen braucht man bei den meisten Programmen. Wir haben bereits den Wert solcher Variablen in unseren grafischen Programmen kennengelernt.

Das Schrittmotorprogramm soll unbegrenzt laufen, daher kommt nun ein Schleifenbaustein mit der Überschrift „Wiederhole unbegrenzt". Die Abbruch-Bedingung wird also hier nie erfüllt. Man verwendet für die Schleife einen unbedingten Sprung.

In der Schleife wird zunächst die „Taste 0" abgefragt. Wenn sie gedrückt ist, so wird in den Ja-Teil gesprungen, wenn nicht, in den Nein-Teil. Dort wird jeweils eine weitere Unterscheidung vorgenommen. Wenn im Ja-Teil der „Taste 0" die „Taste 1" auch zusätzlich gedrückt wurde, dann liegt ein Fehler vor. Denn man kann den Motor nicht zugleich vor und zurück laufen lassen. Es wird dann einfach keine Aktion ausgeführt. Wenn „Taste 1" aber nicht gedrückt ist, und „Taste 0" schon, dann soll der Motor vorwärts laufen. Wenn „Taste 0" nicht gedrückt war, aber „Taste 1", dann soll der Motor rückwärts laufen, und wenn keine Taste gedrückt war, so soll nichts passieren.

Man sieht, daß man mit Hilfe von Struktogrammen besonders schön Fehlerfälle erkennen kann, da sie immer als Spalten von irgendwelchen Abfragen auftreten. Bei einer konventionellen Darstellung mit einem Flußdiagramm wäre vielleicht der Fehlerfall unbemerkt geblieben.

Nun kann man dieses Struktogramm noch nicht direkt in ein Assemblerprogramm übertragen. Man müßte für die Teilprobleme wieder einzelne Struktogramme erstellen, die feiner gegliedert sind.

Wie immer bei so komplexen Aufgaben, so gibt es auch in unserem Fall verschiedene Lösungen. Eine Möglichkeit zeigt das Assemblerprogramm nach der *Abb. 3.2.27.*

Die Pseudobefehle EQU, DC.B
Der Assemblerbefehle MOVEA, LEA, CLR, AND, BEQ

In diesem Programm kommen jetzt eine Reihe von neuen Befehlen vor, mit denen wir uns beschäftigen wollen. Wir sehen uns dazu die Details unserer Lösung an und werden so leicht die Funktionen verstehen.

Wie arbeitet das Programm?

Zunächst folgt der Vereinbarungsbaustein. Diese Unterteilung des Assemblerprogramms in einzelne Bausteine ist ganz willkürlich, sie hat sich jedoch als zweckmäßig erwiesen.

Der Schrittmotor ist über vier Leitungen mit dem Ausgabe-Port verbunden. Es folgt nun die Definition der einzelnen Bits. Dazu wird die „EQU„-Anweisung verwendet. Es handelt sich um eine Anweisung an den Assembler. Der Name, der links neben der „EQU„-Anweisung steht, erhält den Wert der rechten Seite. Also ähnlich wie eine Marke, jedoch ist der Wert frei wählbar. Durch die Verwendung von Namen wird das Programm übersichtlicher. Nochmals in aller Deutlichkeit, es wird hier kein Maschinencode erzeugt, darum auch der Hinweis „Pseudo". Diese Einrichtung dient lediglich der besseren Übersicht.

Sehen wir uns nun für eine sinnvolle Definition unserer symbolischen Namen die Hardware an. Magnet 1 ist an Bit 0 angeschlossen. Der Stellenwert von Bit 0 ist 1; wir müssen also eine Eins ausgeben, um den Magneten 1 zu aktivieren, darum erhält der

```
Rolf-D.Klein 68000/08 Assembler 4.3 (C) 1984, Seite 1

009C00                  ;************************************
009C00                  ; SCHRITTMOTORPROGRAMM VERSION 1.0  ;
009C00                  ; VOM 13.12.1983                    ;
009C00                  ;************************************
009C00
009C00                  ; ******** VEREINBARUNGS-BAUSTEIN ********
009C00                  ;
= 00000001              MAG1    EQU 1   ; MOTORBITS, EINZELNE MAGNETE
= 00000002              MAG2    EQU 2   ; BITZUORDUNGSTABELLE
= 00000004              MAG3    EQU 4
= 00000008              MAG4    EQU 8
009C00
= 00000001              VOR     EQU 1   ; TASTENBIT-ZUORDNUNG
= 00000002              RUECK   EQU 2
009C00
= FFFFFF30              MOTOR   EQU $FFFFFF30
= FFFFFF30              TASTEN  EQU $FFFFFF30
009C00
009C00                  TAB:                    ; SCHALTSEQUENZ-TABELLE
009C00  01                      DC.B MAG1
009C01  03                      DC.B MAG1+MAG2
009C02  02                      DC.B MAG2
009C03  06                      DC.B MAG2+MAG3
009C04  04                      DC.B MAG3
009C05  0C                      DC.B MAG3+MAG4
009C06  08                      DC.B MAG4
009C07  09                      DC.B MAG4+MAG1
009C08
009C08                  ;  ******** UNTERPROGRAMM-BAUSTEIN *********
009C08
009C08                  MOTORVOR:
009C08  5201                    ADDQ.B #1,D1    ; EIN SCHRITT VORWAAERTS
009C0A                                          ; GEMEINSAMER TEIL VOR,RUECK
009C0A  0201 0007       MOTORW: AND.B #7,D1     ; BEREICH 0..7 IST ERLAUBT
009C0E  13F0 1000               MOVE.B 0(A0,D1.W),MOTOR ; AUSGABE AN DEN MOTOR
009C12  FFFFFF30
009C16                                          ; SCHRITTGESCHW. DURCH
009C16  203C 000007D0           MOVE.L #2000,D0 ; WARTESCHLEIFE
009C1C  4E71            WARTE:  NOP             ; FUER ZEITVERZOEGERUNG
009C1E  51C8 FFFC               DBRA D0,WARTE
009C22  4E75                    RTS
009C24
009C24                  MOTORRUECK:
009C24  5301                    SUBQ.B #1,D1    ; EIN SCHRITT RUECKWAERTS
009C26  60E2                    BRA.S MOTORW    ; WEITER WIE OBEN
009C28
009C28                  ; ******** PROGRAMM-BAUSTEIN ************
009C28
009C28  207C 00009C00  START:  MOVEA.L #TAB,A0  ; TABELLENSTART DEFINIEREN
009C2E  323C 0000               MOVE #0,D1      ; MOTOR POSITION ANNEHMEN
009C32  4239 FFFFFF30           CLR.B MOTOR     ; MOTOR AUS
```

zu Abb. 3.2.27

```
Rolf-D.Klein 68000/08 Assembler 4.3 (C) 1984, Seite 2

009C38                                          ; WIEDERHOLE UNBEGRENZT
009C38  1039 FFFFFF30   WDH1:   MOVE.B TASTEN,D0 ; EINLESEN DES PORTS
009C3E  0200 0001               AND.B #VOR,D0   ; VOR-TASTE
009C42  6716                    BEQ.S TASTEVJ   ; KONTAKT SCHLIESST BEI 0
009C44
009C44  1039 FFFFFF30          MOVE.B TASTEN,D0 ; NOCHMALS EINLESEN
009C4A  0200 0002               AND.B #RUECK,D0
009C4E  6622                    BNE.S ENDE      ; NICHT GEDRUECKT
009C50
009C50  4EB9 00009C24          JSR MOTORRUECK  ; MOTOR RUECK
009C56  6000 001A               BRA ENDE
009C5A
009C5A                  TASTEVJ:                ; WENN VORW. DANN
009C5A  1039 FFFFFF30          MOVE.B TASTEN,D0
009C60  0200 0002               AND.B #RUECK,D0 ; NICHT ZUGLEICH RUECKW.
009C64  6700 000C               BEQ ENDE        ; FEHLERFALL
009C68
009C68  4EB9 00009C08          JSR MOTORVOR    ; MOTOR EINEN SCHRITT VOR
009C6E  6000 0002               BRA ENDE
009C72
009C72  6000 FFC4       ENDE:  BRA  WDH1       ; UNBEGRENZT DURCHFUEHREN
009C76
009C76                  ; ***************************************
009C76                  END

008C22  Ende-Symboltabelle
```

Abb. 3.2.27 Assemblerprogramm – Schrittmotor

Name „MAG1" den Wert 1. Magnet 2 liegt an Bit 1, der Stellenwert ist 2, also erhält „MAG2" den Wert 2. Jetzt heißt es aufpassen; um das Bit 3, an dem der Magnet 3 angeschlossen ist, in Aktion zu setzen, müssen wir nicht 3 ausgeben, denn dann würden wir ja die Magnete 1 und 2 gleichzeitig ansteuern, sondern, den Binärzahlen folgend, den Wert 4. „MAG3" erhält also den Wert 4 und „MAG4" den Wert 8. Wenn man den Wert auf den Port ausgibt, so wird der jeweilige Magnet, also die Spule des Schrittmotors eingeschaltet.

Jetzt gilt es, für die Werte, die vom Port für die Tastaturbedienung kommen, übersichtliche Bezeichnungen zu finden. Hier ordnen wir den Namen „VOR" dem Wert 1 und den Namen „RUECK" dem Wert 2 zu. Dies soll die Tastenbelegung an den Ports in Verbindung mit der jeweils ausgelösten Funktion symbolisieren.

Mit „MOTOR EQU $FFFFFF30" wird die Portadresse dem Namen „MOTOR" zugewiesen. Die vielen FFs werden benötigt, um die Portadressen von normalen Speicheradressen zu unterscheiden. Beim 68008 stehen die Portadresse $FFFFFF00 bis $FFFFFFFF zur Verfügung. Beim 68000 sind es doppelt soviele. Will man noch mehr verwenden, so müssen die Peripherieschaltungen geändert werden.

Der Wert $30 am Schluß der FF-Kette ist die Portadresse, die man auch mit den Brücken auf der IOE-Baugruppe eingestellt hat. Hier hatten wir die Codierung 0011, bzw. sedezimal 3 durch die Brückeneinstellung 6 und 7 geschlossen, sowie 4 und 5 offen, vorgenommen.

$30 bis $3F ist der Bereich, bei dem dann die IOE-Baugruppe anspricht. $30 ist die Adresse des Port0, also dort, wo im Falle einer Ausgabe der Schrittmotor angeschlossen ist und im Falle einer Eingabe das Tastenfeld. Daher wird auch mit „TASTEN EQU $FFFFFF30" der Eingabeport definiert.

Aufgabe:

1. Prüfen Sie mit Hilfe des Menues IO-Lesen die Funktion des Tastenfeldes. Dazu wird die Adresse $FFFFFF30 eingeben. Dann wird die Taste „D" auf der Tastatur gedrückt, um eine dauernde Abfrage zu ermöglichen. Wenn man nun eine der Einzeltasten des zusätzlichen Tastenfeldes drückt, so muß das jeweilige Bit auf dem Bildschirm eine 0 zeigen.
2. Prüfen Sie die Schrittmotorschaltung. Dazu ruft man das Menue IO-Setzen auf. Nun kann man den Motor bewegen, wenn man nacheinander die richtigen Bitkombinationen auf den Port $FFFFFF30 ausgibt. Es ist dies die Sequenz: 1,3,2,6,4,$C,8,9,1,3,2, 6,4,.....

   Es wird hier das Halbschrittverfahren verwendet. Das bedeutet, daß manchmal auch nur eine Spule vom Strom durchflossen wird. Dadurch kann man den Motor aber sehr feinstufig bewegen.

Im Programm wird für die einzelnen Werte eine Tabelle definiert. Diese beginnt bei der Marke „TAB". Der Befehl „DC.B" ist ebenfalls eine Anweisung an den Assembler. Die Werte, die rechts neben dieser Anweisung stehen, werden im Speicher abgelegt. Das ".B" besagt, daß die Werte als Bytegrößen im Speicher abgelegt werden sollen.

So legt der Befehl „DC.B MAG1" den Wert 01 im Speicher ab. Mit „DC.B MAG 1+MAG2" wird der Wert 3 abgelegt, denn die Addition der Werte von „MAG1" und „MAG2" ergibt den Wert 3. Man hätte auch „DC.B 3" schreiben können, jedoch ist die hier gewählte Darstellung verständlicher. Auf diese Weise werden alle 8 Schritte eingetragen. Man kann übrigens beim „DC„-Befehl auch mehrere Werte, durch Komma getrennt, in eine Zeile schreiben.

Nun muß man mit neuen Befehlen auf diese Tabelle zugreifen. Ziel ist es, jeweils einen Wert aus der Tabelle zu holen und an den Port auszugeben. Beim Vorwärtslauf soll dabei nacheinander Wert für Wert beginnend bei der ersten Stelle bis zur letzten ausgegeben werden. Wenn man alle 8 Werte ausgegeben hat, soll dann wieder mit dem Anfang der Tabelle begonnen werden. Beim Rückwärtslauf verfährt man umgekehrt. Die Tabelle wird rückwärts, also von unten nach oben ausgegeben.

Dazu betrachten wir das Hauptprogramm, das bei der Marke „START" beginnt. Der erste Befehl lautet: „MOVEA.L #TAB,A0" und bedeutet: Lade den Wert von „TAB", also die Adresse der Tabelle, in das Adreßregister A0. Das „A" hinter „MOVE" besagt, daß man etwas in ein Adreßregister laden will. Es gibt noch einen zweiten Befehl, der das gleiche bewirken kann: „LEA TAB,A0", er bedeutet: Lade effektive Adresse nach A0. Der feine Unterschied bei den zugelassenen Adreßierarten soll uns hier noch nicht interessieren.

Im Register A0 steht also die Adresse der Tabelle „TAB".

In das Register D1 wird nun der Wert 0 geladen. Das Register D1 soll nämlich die Position innerhalb der Tabelle bestimmen. Der Befehl „CLR.B MOTOR" sendet zur Adresse MOTOR ein Nullbyte und schaltet somit den Motor ab. Alle Spulen sind dann stromlos.

Nun beginnt die Schleife bei „WDH1". Mit „MOVE.B TASTEN,D0" wird das Tastenfeld in das Register D0.B geladen. Nun kann man mit einem Befehl „AND.B #VOR,D0" ein einzelnes Bit prüfen, und zwar Bit 0. „AND" bedeutet UND und ist ein Befehl zur Und-Verknüpfung. Der Inhalt von D0.B wird also mit 1 Und-Verknüpft. Wenn das Ergebnis 0 ist, so wurde die Taste gedrückt, sonst nicht. Mit einem „BEQ„-Befehl kann man also prüfen, ob die Taste gedrückt wurde oder nicht. Das ".S" hinter dem Befehl „BEQ" steht für „SHORT", dadurch wird der Code für den Befehl kürzer, aber auch der mögliche Sprungabstand.

Auf die gleiche Weise wird auch die Taste „RUECK" geprüft. Mit „JSR MOTORRUECK" und „JSR MOTORVOR" wird das Unterprogramm zur Motorsteuerung aufgerufen. Dieses Unterprogramm ist im Unterprogrammbaustein definiert.

Bei der Marke „MOTORVOR" wird der Wert 1 auf den Inhalt von D1.B addiert. Dadurch soll der nächste Eintrag der Tabelle ermittelt werden. Wenn nun ein Wert größer als 8 auftritt, so muß aber wieder beim Tabellenanfang begonnen werden. Dazu wird anschließend bei der Marke „MOTORW" mit „AND.B #7,D1" eine Und-Verknüpfung mit dem Wert 7 vorgenommen. D1 kann jetzt nur noch die Werte 0,1,2,3,4,5,6,7 annehmen. Dann folgt ein neuer, sehr kompliziert aussehender Befehl:
„MOVE.B 0(A0,D1.W),MOTOR"

Der Befehl bedeutet: Nimm den Inhalt des Registers A0, also die Adresse der Tabelle, addiere dazu den Inhalt des Registers D1.W, also einen Wert von 0 bis 7. Dann addiere den Wert 0 dazu (die 0 vor der Klammer). Das ganze verwende als neue Adresse, und hole von dort ein Byte (da .B hinter MOVE steht). Dieses Byte gib an die Adresse „MOTOR", also an den Port des Schrittmotors aus. *Abb. 3.2.28* zeigt eine Übersicht aller Adressierarten. Diese Vielzahl muß man aber keinesfalls sofort nutzen können.

Jetzt wieder weiter in unserem Programm. Nach der Ausgabe an die Adresse „MOTOR" folgt eine sogenannte Warteschleife. Da der Motor nicht beliebig schnell laufen kann, muß man nach jedem ausgeführten Schritt eine kleine Ruhepause von einigen Millisekunden einlegen. Dazu wird das Register D0 mit einem Wert geladen. Dann wird ein „NOP„-Befehl ausgeführt und mit dem „DBRA„-Befehl der Inhalt von D0 bei jedem Schleifendurchlauf um 1 verringert. Da die Befehlsausführung von „NOP" und „DBRA" eine bestimmte Zeit braucht, entsteht eine Verzögerung.

Bei der Ausführung von „MOTORRUECK" wird umgekehrt verfahren. Der Inhalt von D1 wird um eins verringert, dann folgt ein Sprung zur Marke „MOTORW", wo wie bei „MOTORVOR" verfahren wird. Durch den Und-Befehl wird auch hier erreicht, daß der Inhalt von D1 nie einen anderen Wert als 0 bis 7 annimmt.

Aufgaben:

1. Zeichnen Sie ein Struktogramm für das Unterprogramm „MOTORVOR".
2. Warum wird die Warteschleife 2001 mal durchlaufen, obwohl der Wert 2000 in das Register D0 geladen wurde?
3. Was bewirkt das Attribut ".S" bei den Befehlen „BEQ", „BRA", „BNE" ?
4. Zeichnen Sie ein Struktogramm für die Warteschleife.
5. Was passiert, wenn man bei der „ADD" und „SUB" Anweisung im Unterprogrammbaustein den Wert 2 anstelle von 1 einsetzt? Ausprobieren!
6. Wie kann man den Schrittmotor langsamer oder schneller laufen lassen?
7. Wie kann man die Einzeltastenabfrage durch Tastaturabfragen ersetzen? (Hinweis: man benötigt das Unterprogramm CSTS und CI).

3.3 Anschluß eines Plotters

Im vorherigen Abschnitt haben wir Schrittmotoren kennengelernt. Hier soll nun eine praktische Anwendung damit behandelt werden: Die Ansteuerung eines Plotters, eines elektronischen Zeichentisches also.

Schrittmotoren eignen sich hervorragend zum Bau von Plottern. In der Praxis unterscheidet man zwei verschiedene Grundtypen: den Flachbettplotter und den Trommelplotter.

Beim Flachbettplotter liegt das Papier fest auf einer Unterlage, und der Stift wird in X- und Y-Richtung über das Papier bewegt.

Syntax	Beispiel	Name
Dn	D0	Datenregister
An	A3	Adressregister
(An)	(A2)	Indirekt
(An)+	(A7)+	Postincrement
-(An)	-(A7)	Predecrement
d(An)	56(A0)	Displacement
d(An,Xn)	45(A0,D4.L)	Index mit Displ.
Abs.W	$1234.W	Absolut, kurz
Abs.L	$88991122.L	Absolut, lang
d(PC)	START(PC)	PC-Relativ
d(PC,Xn)	GEHT(PC,A3.W)	PC-Relativ, Index
Imm	#5422	Direkt

Abb. 3.2.28 Adressierarten

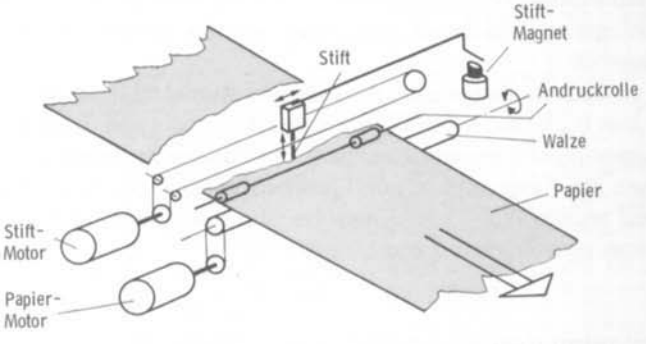


Abb. 3.3.1 Schematischer Aufbau des Selbstbau-Plotters

Eine andere Art sind die Trommelplotter. Das Papier ist dabei in eine Walze eingespannt und bewegt sich z.B. in Y-Richtung. Der Stift bewegt sich nur in X-Richtung. Auch dabei kann man jeden Punkt des Papiers genau anwählen.

Der Trommelplotter hat den Vorteil, daß man kleinere Massen bewegen muß, nämlich einmal nur das Papier durch die Walze und zum anderen nur den Stift. Beim Flachbettplotter muß man auch noch eine Achse bewegen, auf der der Stift läuft.

Daher wollen wir einen Trommelplotter aufbauen. Dazu gibt es im Handel einen fertigen Bausatz, der alle mechanischen Teile enthält, sowie ein eigenes Netzteil und die notwendigen Treibertransistoren etc. Man kann diesen Plotter einfach an einen Parallelport, also z. B. die IOE-Baugruppe, anschließen. Die *Abb. 3.3.1* zeigt ein Schema der Arbeitsweise. Es gibt einen Schrittmotor für den Papiertransport und einen für den Stift. Dabei ist die mechanische Konstruktion hier etwas vereinfacht dargestellt. Ein zusätzlicher Stiftmagnet sorgt dafür, daß man den Schreibstift wahlweise heben oder senken kann. Der Plotter kann eine Genauigkeit von ca. 0.1mm erreichen.

Die *Abb. 3.3.2* zeigt das Anschlußschema. Man benötigt neun Port-Ausgänge, um den Plotter anzusteuern. Dabei dienen je vier Leitungen der Ansteuerung der beiden Schrittmotoren und eine Leitung ist für den Stiftmagnet gedacht. Wenn die Leitungen ein 1-Signal führen, so ist die betreffende Spule stromlos. Achtung! Das ist genau umgekehrt wie bei der Schaltung aus dem vorherigen Kapitel.

Der Plotter wird über ein Flachbandkabel mit der IOE-Baugruppe verbunden. Dazu kann man sich eine entsprechende Stiftleiste auf der IOE-Baugruppe auflöten, wie es die *Abb. 3.3.3* zeigt.

Wie man Schrittmotoren ansteuert, wissen wir schon. Hier kommen aber zwei Probleme neu hinzu. Zum einen müssen zwei Schrittmotoren bedient werden, und zum anderen soll ein Plotter fest vorgegebene Linien zeichnen können. Man muß jetzt also die Schrittzahl genau berechnen und nicht wie im letzten Abschnitt einfach per Tastenfeld eingeben.

Dazu ist es am besten, die gestellte Aufgabe erst einmal zu reduzieren. Wir wollen die Ausgabe auf dem Bildschirm durchführen. Die *Abb. 3.3.4* soll dargestellt werden. Wie muß das dazugehörige Programm aussehen? Die Fortgeschrittenen unter Ihnen können vielleicht schon ein Programm in Schildkrötensprache dazu erzeugen, wir wollen es aber einmal auf andere Weise versuchen. Im Grundprogramm gibt es zwei elementare Unterprogramme zum Zeichnen von Linien:

Die Grundprogrammbefehle MOVETO und DRAWTO

Der Befehl „MOVETO" positioniert den Schreibstift des Graphik-Prozessors auf die Koordinaten X=D1.W und Y=D2.W. Der Aufruf „DRAWTO" zeichnet dann eine Linie zum Ziel mit X=D1.W und Y=D2.W.

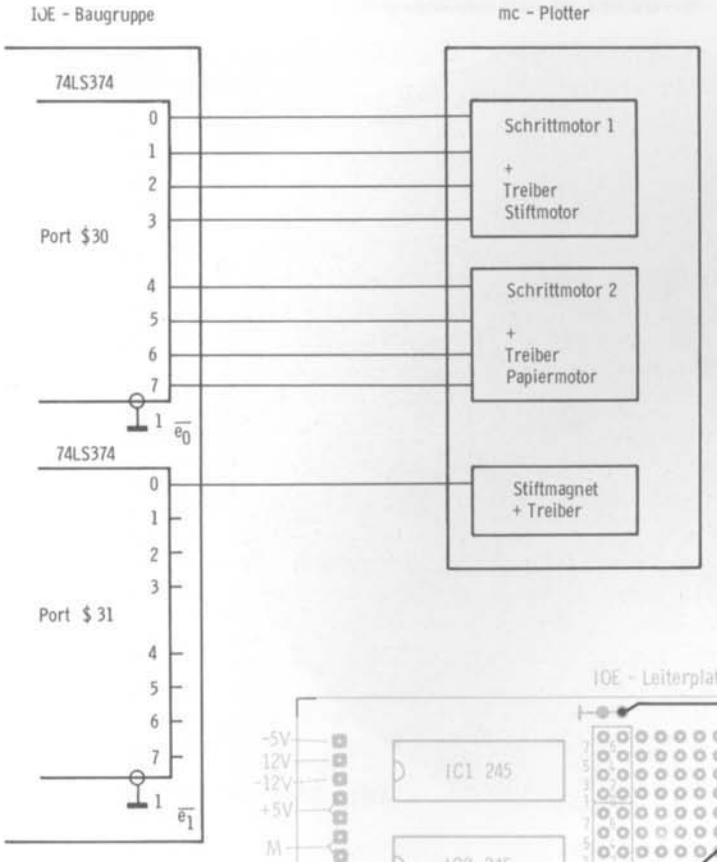


Abb. 3.3.2 Anschlußschema für die IOE-Baugruppe

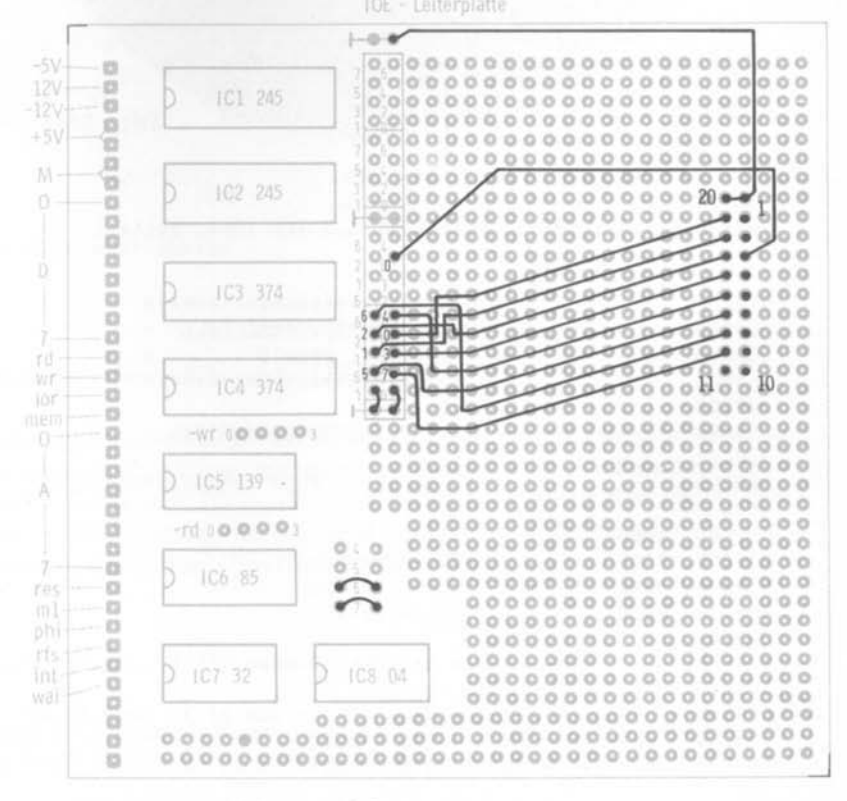


Abb. 3.3.3 Die Verdrahtung auf der IOE- Baugruppe

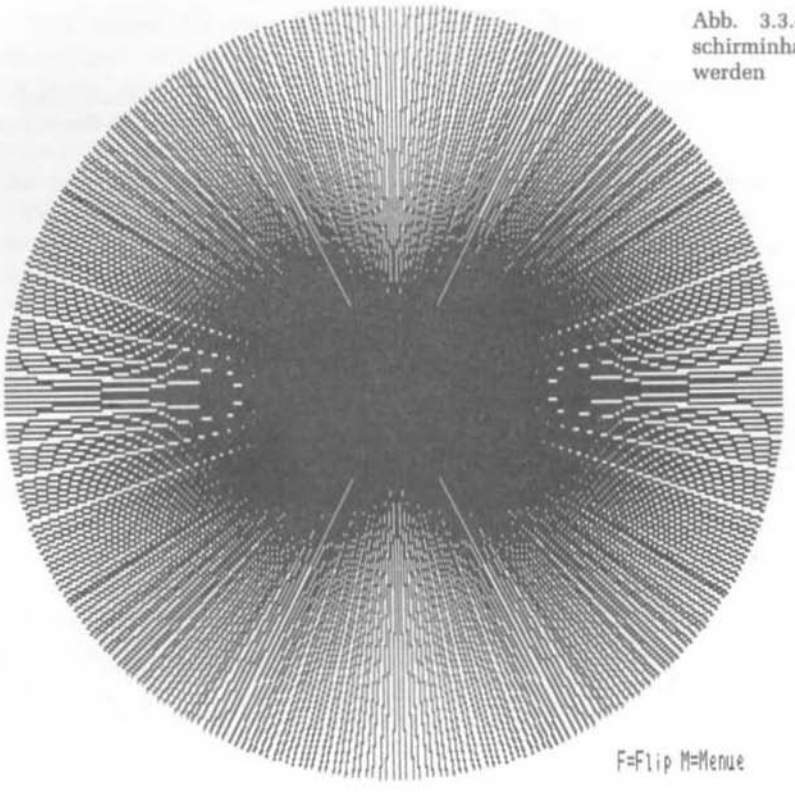


Abb. 3.3.4 Dieser Bildschirminhalt soll dargestellt werden

```
Rolf-D.Klein 68000/08 Assembler 4.3 (C) 1984, Seite 1

009C00                  ************************
009C00                  * GRUNDPROGRAMMBEFEHLE  *
009C00                  * MOVETO, DRAWTO        *
009C00                  ************************
009C00
009C00                  * VEREINBARUNGSBAUSTEIN
009C00
009C00                  * VARIABLE FUER PARAMETERUEBERGABE
009C00
009C00  0000            X1: DC.W 0
009C02  0000            Y1: DC.W 0
009C04  0190            X2: DC.W 400
009C06  0078            Y2: DC.W 120
009C08
009C08                  * UNTERPROGRAMMBAUSTEIN
009C08
009C08                  LINIE:      * VON X1,Y1 NACH X2,Y2
009C08  3239 00009C00    MOVE X1,D1
009C0E  3439 00009C02    MOVE Y1,D2
```

zu Abb. 3.3.5

```
009C14  4EB9 00000EE2     JSR @MOVETO
009C1A  3239 00009C04     MOVE X2,D1
009C20  3439 00009C06     MOVE Y2,D2
009C26  4EB9 00000F4C     JSR @DRAWTO
009C2C  4E75              RTS
009C2E
009C2E                  * PROGRAMMBAUSTEIN
009C2E
009C2E                  START:
009C2E  383C 0168         MOVE #360,D4    * LINIEN IN VERSCHIEDENEN
009C32                  LPP:              * RICHTUNGEN ZEICHNEN, ALS
009C32  33FC 0100         MOVE #256,X1    * TEST
009C36  00009C00
009C3A  33FC 0080         MOVE #128,Y1    * X1,Y1 IST MITTELPUNKT
009C3E  00009C02
009C42  3004              MOVE D4,D0      * X2=SIN256(PHI)+256
009C44  4EB9 00001034     JSR @SIN        * VON MITTELPUNKT AUS
009C4A  0640 0100         ADD #256,D0     * AUF KREISBAHNEN
009C4E  33C0 00009C04     MOVE D0,X2      *
009C54  3004              MOVE D4,D0      * Y2=COS256(PHI)/2+128
009C56  4EB9 00001030     JSR @COS
009C5C  48C0              EXT.L D0
009C5E  81FC 0002         DIVS #2,D0
009C62  0640 0080         ADD #128,D0
009C66  33C0 00009C06     MOVE D0,Y2
009C6C  2F04              MOVE.L D4,-(A7)   * D4 RETTEN
009C6E  6100 FF98         BSR LINIE         * BRESENHAM AUFRUFEN
009C72  281F              MOVE.L (A7)+,D4   * D4 WIEDER ZURUECK
009C74  51CC FFBC         DBRA D4,LPP
009C78  4E75              RTS
009C7A
009C7A
009C7A                  END

008C58  Ende-Symboltabelle
```

Abb. 3.3.5 Programm mit den Grundprogrammbefehlen: „MOVETO" und „DRAWTO"

Die Grundprogrammbefehle SIN und COS

Um den Fächer mathematisch zu konstruieren, braucht man noch die Unterprogramme „SIN" und „COS", die im Grundprogramm bereits vorhanden sind. Als Eingabe verlangen sie einen Winkel im Register D0.W; als Ergebnis liefern sie einen Wert von +/-256 ebenfalls im Register D0.W. Mathematisch ist es also nicht der Sinus, oder Cosinus, sondern ein 256faches des Wertes ohne Nachkommastellen.

Der Pseudobefehl DC.W

Die Abb. 3.3.5 zeigt die Lösung als Programm. Das Unterprogramm LINIE zeichnet eine Linie von X1;Y1 nach X2;Y2. X1,Y1,X2,Y2 sind dabei Speicherzellen, in denen die

Koordinaten stehen. Diese Speicherzellen werden im Vereinbarungsteil unseres Programms mit festen Werten voreingestellt. Dazu verwenden wir den neuen Befehl DC.W. Hier handelt es sich wie beim bereits bekannten Befehl DC.B um einen Pseudobefehl, der Unterschied liegt nur am .W, es werden also Worte und nicht Bytes abgelegt. Mit DC.W 0 wird der Wert 0 an die aktuelle Speicherzelle geschrieben. Die Marke davor gibt der Speicherzelle einen Namen, also z. B. „X1". Mit einem „MOVE X1,D0" kann man dann den Wert in das Register D0 laden. Mit „MOVE D0,X1" kann man den Inhalt von D0 zur Speicherzelle „X1" transportieren.

Nach dem Starten des Programmes ergibt sich das gewünschte Bild.

Wenn man das Programm auf den Plotter übertragen will, so muß man allerdings noch einiges tun. Das Hauptproblem ist, eine Linie zu zeichnen. Der Plotter kann immer nur einen kleinen Schritt ausführen. Der Computer muß also den genauen Verlauf des Weges berechnen. Die Aufgabe, zwei Punkte durch eine Linie miteinander zu verbinden, leistet bei uns das Grundprogramm mit der Routine „DRAWTO". Dabei wird das Problem nochmals auf den Graphikprozessor selbst übertragen, der den Algorithmus in seinem Inneren ausführt.

Glücklicherweise ist bekannt, wie das funktioniert. Grundlage dafür bildet der sogenannte Bresenham-Algorithmus, der für Plotter und Graphikbildschirm von einem Herrn Bresenham entwickelt wurde.

Der Algorithmus ist besonders schnell und kommt ohne Multiplikation und Division aus, die man normalerweise benötigen würde, um die Steigung der Geraden durch die beiden Punkte zu ermitteln. Die *Abb. 3.3.6* zeigt die komplette Lösung. Nun wird nur noch das Unterprogramm MOVETO verwendet, um die Position des Bildpunktes festzulegen. Mit dem Unterprogramm „CMD" und dem Wert $80 kann dann ein Bildpunkt gezielt gesetzt werden.

Aufgabe:

1. Versuchen Sie, ein Struktogramm zum Bresenhamalgorithmus zu entwickeln.
2. Versuchen Sie mit einem kleinen Beispiel, den Algortithmus auf einem Stück Papier nachzuvollziehen.

```
Rolf-D.Klein 68000/08 Assembler 4.3 (C) 1984, Seite 1


009C00                  **************************
009C00                  * BRESENHAM ALGORITHMUS *
009C00                  * ROLF-DIETER KLEIN     *
009C00                  **************************
009C00
009C00                  * VEREINBARUNGSBAUSTEIN
009C00
009C00                  * VARIABLE FUER PARAMETERUEBERGABE
009C00
009C00  0000            X1: DC.W 0
009C02  0000            Y1: DC.W 0              zu Abb. 3.3.6
```

```
009C04  0190            X2: DC.W 400
009C06  0078            Y2: DC.W 120
009C08
009C08                  * UNTERPROGRAMMBAUSTEIN
009C08
009C08                  LINIE:      * VON X1,Y1 NACH X2,Y2
009C08  383C 0001        MOVE #1,D4 * RICHTUNG X
009C0C  3A39 00009C04    MOVE X2,D5
009C12  9A79 00009C00    SUB X1,D5  * DX BESTIMMEN
009C18  6A00 0008        BPL ST1
009C1C  383C FFFF        MOVE #-1,D4
009C20  4445             NEG D5
009C22                  ST1:
009C22  3C3C 0001        MOVE #1,D6 * RICHTUNG Y
009C26  3E39 00009C06    MOVE Y2,D7
009C2C  9E79 00009C02    SUB Y1,D7  * DY BESTIMMEN
009C32  6A00 0008        BPL ST2
009C36  3C3C FFFF        MOVE #-1,D6
009C3A  4447             NEG D7
009C3C                  ST2:
009C3C  3605             MOVE D5,D3  * S=DX
009C3E  BA47             CMP D7,D5   * DX < DY DANN SPEZ
009C40  6A00 0006        BPL ST3
009C44  3607             MOVE D7,D3
009C46  4443             NEG D3      * S=-DY
009C48                  ST3:
009C48                  S4:
009C48  3239 00009C00    MOVE X1,D1
009C4E  3439 00009C02    MOVE Y1,D2
009C54  4EB9 00000EE2    JSR @MOVETO  * POSITION SETZEN
009C5A  103C 0080        MOVE.B #$80,D0  * PUNKT-BEFEHL
009C5E  4EB9 00000D66    JSR @CMD     * DORTHIN PUNKT SETZEN
009C64  3039 00009C00    MOVE X1,D0
009C6A  B079 00009C04    CMP X2,D0
009C70  6700 0028        BEQ S7
009C74                  S5:
009C74  4A43             TST D3
009C76  6B00 0012        BMI S6
009C7A  D244             ADD D4,D1
009C7C  33C1 00009C00    MOVE D1,X1
009C82  9647             SUB D7,D3
009C84  9647             SUB D7,D3
009C86  6000 FFC0        BRA S4
009C8A                  S6:
009C8A  D446             ADD D6,D2
009C8C  33C2 00009C02    MOVE D2,Y1
009C92  D645             ADD D5,D3
009C94  D645             ADD D5,D3
009C96  6000 FFB0        BRA S4
009C9A                  S7:
009C9A  3039 00009C02    MOVE Y1,D0
009CA0  B079 00009C06    CMP Y2,D0
```

zu Abb. 3.3.6

```
Rolf-D.Klein 68000/08 Assembler 4.3 (C) 1984, Seite 2

009CA6  6600 FFCC          BNE S5
009CAA  4E75               RTS
009CAC
009CAC                     * PROGRAMMBAUSTEIN
009CAC
009CAC                     START:
009CAC  383C 0168          MOVE #360,D4    * LINIEN IN VERSCHIEDENEN
009CB0                     LPP:            * RICHTUNGEN ZEICHNEN, ALS
009CB0  33FC 0100          MOVE #256,X1    * TEST
009CB4  00009C00
009CB8  33FC 0080          MOVE #128,Y1    * X1,Y1 IST MITTELPUNKT
009CBC  00009C02
009CC0  3004               MOVE D4,D0      * X2=SIN256(PHI)+256
009CC2  4EB9 00001034      JSR @SIN        * VON MITTELPUNKT AUS
009CC8  0640 0100          ADD #256,D0     * AUF KREISBAHNEN
009CCC  33C0 00009C04      MOVE D0,X2      *
009CD2  3004               MOVE D4,D0      * Y2=COS256(PHI)/2+128
009CD4  4EB9 00001030      JSR @COS
009CDA  48C0               EXT.L D0
009CDC  81FC 0002          DIVS #2,D0
009CE0  0640 0080          ADD #128,D0
009CE4  33C0 00009C06      MOVE D0,Y2
009CEA  2F04               MOVE.L D4,-(A7)  * D4 RETTEN
009CEC  6100 FF1A          BSR LINIE        * BRESENHAM AUFRUFEN
009CF0  281F               MOVE.L (A7)+,D4  * D4 WIEDER ZURUECK
009CF2  51CC FFBC          DBRA D4,LPP
009CF6  4E75               RTS
009CF8
009CF8
009CF8                     END

008C58  Ende-Symboltabelle
```

Abb. 3.3.6 Der Bresenham-Algorithmus

Dieser Algorithmus ist in dem Graphikprozessor GDP EF9366 direkt und fest eingebaut. Daher zeichnet unser Algorithmus die Linien auch in gleicher Weise.

Diesen Bresenham-Algorithmus kann man nun verwenden, um den Plotter zu bedienen. Denn es wird maximal um einen Punkt in X-Richtung und/oder in Y-Richtung vorangeschritten.

Die *Abb. 3.3.7* zeigt die komplette Lösung. Das Hauptprogramm steuert den Plotter so,

```
Rolf-D.Klein 68000/08 Assembler 4.3 (C) 1984, Seite 1

009C00                     *************************
009C00                     * PLOTTERANSCHLUSS      *
009C00                     * VERSION 1.0           *
009C00                     * ROLF-DIETER KLEIN     *
009C00                     *************************
009C00
```

zu Abb. 3.3.7

```
009C00                  * ACHTUNG: PROGRAMM AUF $A000
009C00                  * EINGEBEN DA SEHR LANG
009C00                  * VEREINBARUNGSBAUSTEIN
009C00
009400                   ORG $9400      * DORTHIN OBJEKTCODE
009400
= 00000001              MAG1 EQU 1      * PORT BIT 0,4
= 00000002              MAG2 EQU 2      * PORT BIT 1,5
= 00000004              MAG3 EQU 4      * PORT BIT 2,6
= 00000008              MAG4 EQU 8      * PORT BIT 3,7
009400
= FFFFFF30              MOTOR EQU $FFFFFF30 * SCHRITTMOTORE
= FFFFFF31              STIFT EQU $FFFFFF31 * STIFTMAGNET BIT 0
009400
009400                  TAB:            * FUER VOLLSCHRITT
009400  03               DC.B MAG1+MAG2
009401  06               DC.B MAG2+MAG3
009402  0C               DC.B MAG3+MAG4
009403  09               DC.B MAG4+MAG1
009404
009404                  * VARIABLE FUER PARAMETERUEBERGABE
009404
009404  0000            STEP1: DC.W 0   * MOTOR 1 SCHRITTLAGE
009406  0000            STEP2: DC.W 0   * MOTOR 2 SCHRITTLAGE
009408
009408  0000            X0: DC.W 0      * ALTE POSITION
00940A  0000            Y0: DC.W 0      * DES MOTORS
00940C  0000            X1: DC.W 0      * NEUE POSITION
00940E  0000            Y1: DC.W 0
009410  0000            X2: DC.W 0
009412  0000            Y2: DC.W 0      * DES MOTORS
009414
009414  0000            XA: DC.W 0      * PARAMETER FUER MLINIE
009416  0000            YA: DC.W 0      *
009418  0000            XE: DC.W 0
00941A  0000            YE: DC.W 0
00941C
00941C                  * UNTERPROGRAMMBAUSTEIN
00941C
00941C                  HEBE:       * STIFT HOCH
00941C  13FC 0001        MOVE.B #1,STIFT
009420  FFFFFF31
009424  6100 0012        BSR WARTE1 * WARTEN BIS WIRKLICH OBEN
009428  4E75             RTS
00942A
00942A                  SENKE:      * STIFT RUNTER
00942A  13FC 0000        MOVE.B #0,STIFT
00942E  FFFFFF31
009432  6100 0004        BSR WARTE1 * WARTEN BIS GESENKT
009436  4E75             RTS
009438
009438                  WARTE1:
009438  303C 1388        MOVE #5000,D0
```

zu Abb. 3.3.7

```
Rolf-D.Klein 68000/08 Assembler 4.3 (C) 1984, Seite 2

00943C  6000 0032         BRA WARTE    * VERZOEGERUNG FUER STIFT
009440                    *
009440                    STEPAUSF:    * STEP1,STEP2 ALS PARAMETER
009440  41F9 00009400      LEA TAB,A0   * BERECHNEN DES CODES
009446  3039 00009406      MOVE STEP2,D0
00944C  0240 0003          AND #3,D0    * GARANTIERT IM BEREICH
009450  1230 0000          MOVE.B 0(A0,D0.W),D1 * LSB TEIL
009454  E919               ROL.B #4,D1 * VIER BITS NACH LINKS
009456  3039 00009404      MOVE STEP1,D0
00945C  4440               NEG D0       * DAMIT X,Y - ORIENTIERUNG OK
00945E  0240 0003          AND #3,D0    * BEREICH 0..3
009462  8230 0000          OR.B 0(A0,D0.W),D1 * UND DAZU NEHMEN
009466  13C1 FFFFFF30      MOVE.B D1,MOTOR * SCHRITT AUSFUEHREN
00946C  303C 07D0          MOVE #2000,D0    * DANN WARTEN
009470                    WARTE:
009470  4E71               NOP
009472  51C8 FFFC          DBRA D0,WARTE    * FUER SCHRITTMOTOR
009476  4E75               RTS
009478
009478                    MLINIE:      * LINIE VON XA,YA, NACH XE,YE
009478  3239 00009414      MOVE XA,D1
00947E  3439 00009416      MOVE YA,D2
009484  6100 0014          BSR MOVETO
009488  3239 0000941B      MOVE XE,D1
00948E  3439 0000941A      MOVE YE,D2
009494  6100 003A          BSR DRAWTO
009498  4E75               RTS
00949A
00949A                    MOVETO:          * D1 = X, D2 = Y
00949A  6100 FF80          BSR HEBE        * ZUNAECHST OHNE STIFT
00949E  33F9 00009408      MOVE X0,X1
0094A4  0000940C
0094A8  33F9 0000940A      MOVE Y0,Y1      * ANFANGSPUNKT
0094AE  0000940E
0094B2  33C1 00009410      MOVE D1,X2
0094B8  33C2 00009412      MOVE D2,Y2      * ENDPUNKT
0094BE  33C1 00009408      MOVE D1,X0      * NEUER ENDPUNKT
0094C4  33C2 0000940A      MOVE D2,Y0
0094CA  6100 003A          BSR LINIE       * UND ANFAHREN
0094CE  4E75               RTS
0094D0
0094D0                    DRAWTO:          * D1 = X, D2 = Y
0094D0  6100 FF58          BSR SENKE       * UND ZEICHNEN
0094D4  33F9 00009408      MOVE X0,X1      * NEUER ANFANGSPUNKT
0094DA  0000940C
0094DE  33F9 0000940A      MOVE Y0,Y1
0094E4  0000940E
0094E8  33C1 00009410      MOVE D1,X2      * NEUER ENDPUNKT
0094EE  33C2 00009412      MOVE D2,Y2
0094F4  33C1 00009408      MOVE D1,X0      *
0094FA  33C2 0000940A      MOVE D2,Y0
```

zu Abb. 3.3.7

```
Rolf-D.Klein 68000/08 Assembler 4.3 (C) 1984, Seite 3

009500  6100 0004        BSR LINIE
009504  4E75             RTS
009506
009506
009506                                * NACH BRESENHAM
009506                   LINIE:       * VON X1,Y1 NACH X2,Y2
009506
009506
009506  383C 0001        MOVE #1,D4 * RICHTUNG X
00950A  3A39 00009410    MOVE X2,D5
009510  9A79 0000940C    SUB X1,D5  * DX BESTIMMEN
009516  6A00 0008        BPL ST1
00951A  383C FFFF        MOVE #-1,D4
00951E  4445             NEG D5
009520                   ST1:
009520  3C3C 0001        MOVE #1,D6 * RICHTUNG Y
009524  3E39 00009412    MOVE Y2,D7
00952A  9E79 0000940E    SUB Y1,D7  * DY BESTIMMEN
009530  6A00 0008        BPL ST2
009534  3C3C FFFF        MOVE #-1,D6
009538  4447             NEG D7
00953A                   ST2:
00953A  3605             MOVE D5,D3  * S=DX
00953C  BA47             CMP D7,D5   * DX < DY DANN SPEZ
00953E  6A00 0006        BPL ST3
009542  3607             MOVE D7,D3
009544  4443             NEG D3      * S=-DY
009546                   ST3:
009546                   S4:
009546  6100 FEF8        BSR STEPAUSF * UND AUSFUEHREN
00954A  3239 0000940C    MOVE X1,D1
009550  3439 0000940E    MOVE Y1,D2
009556  3039 0000940C    MOVE X1,D0
00955C  B079 00009410    CMP X2,D0
009562  6700 0034        BEQ S7
009566                   S5:
009566  4A43             TST D3
009568  6B00 0018        BMI S6
00956C  D979 00009404    ADD D4,STEP1 * X-RICHTUNG
009572  D244             ADD D4,D1
009574  33C1 0000940C    MOVE D1,X1
00957A  9647             SUB D7,D3
00957C  9647             SUB D7,D3
00957E  6000 FFC6        BRA S4
009582                   S6:
009582  D446             ADD D6,D2
009584  DD79 00009406    ADD D6,STEP2 * Y-RICHTUNG
00958A  33C2 0000940E    MOVE D2,Y1
009590  D645             ADD D5,D3
009592  D645             ADD D5,D3
009594  6000 FFB0        BRA S4
009598                   S7:
```

zu Abb. 3.3.7

```
Rolf-D.Klein 68000/08 Assembler 4.3 (C) 1984, Seite 4

009598  3039 0000940E    MOVE Y1,D0
00959E  B079 00009412    CMP Y2,D0
0095A4  6600 FFC0        BNE S5
0095A8  4E75             RTS
0095AA
0095AA                   * PROGRAMMBAUSTEIN
0095AA
0095AA                   START:
0095AA  4279 00009414    CLR XA
0095B0  33FC FE00        MOVE #-512,YA
0095B4  00009416
0095B8  4279 00009418    CLR XE
0095BE  33FC 0200        MOVE #512,YE
0095C2  0000941A
0095C6  6100 FEB0        BSR MLINIE
0095CA  4279 00009416    CLR YA
0095D0  4279 0000941A    CLR YE
0095D6  4279 00009414    CLR XA
0095DC  33FC 04B0        MOVE #1200,XE
0095E0  00009418
0095E4  6100 FE92        BSR MLINIE
0095E8                   *
0095E8  4241             CLR D1
0095EA  4242             CLR D2
0095EC  4EB9 00000EE2    JSR @MOVETO
0095F2  6100 FEA6        BSR MOVETO   * AUF DEN 0-PUNKT
0095F6  4244             CLR D4   * STARTWINKEL
0095F8                   SCHLEIFE:
0095F8  3004             MOVE D4,D0 *
0095FA  48C0             EXT.L D0   * VORBEREITEN FUER DIVISION
0095FC  81FC 0002        DIVS #2,D0
009600  4EB9 00001034    JSR @SIN
009606  3400             MOVE D0,D2 *
009608  3004             MOVE D4,D0
00960A  C1FC 0007        MULS #7,D0
00960E  4EB9 00001030    JSR @COS
009614  48C0             EXT.L D0   * FUER DIV VORBEREITEN
009616  81FC 0004        DIVS #4,D0
00961A  C5C0             MULS D0,D2  * SIN256(PHI/2)*COS256(PHI*7)/4
00961C  85FC 0020        DIVS #32,D2 * ANGLEICH FUER Y-KOORDINATE
009620  3204             MOVE D4,D1  * WINKEL IST X-KOORDINATE
009622  3F04             MOVE D4,-(A7) * MERKEN
009624  6100 FEAA        BSR DRAWTO
009628  381F             MOVE (A7)+,D4 * ZURUECK
00962A  0644 0001        ADD #1,D4     * UM JE 1 GRAD
00962E  0C44 04B0        CMP #1200,D4  * BIS ENDE
009632  6500 FFC4        BCS SCHLEIFE  * BIS GROESSER
009636  4241             CLR D1
009638  4242             CLR D2
00963A  6100 FE5E        BSR MOVETO * ZUM NULLPUNKT, STIFT HOCH
00963E  4E75             RTS
009640
009640
009640                   END

008C4D  Ende-Symboltabelle
```

Abb. 3.3.7 Das Plotter-Programm

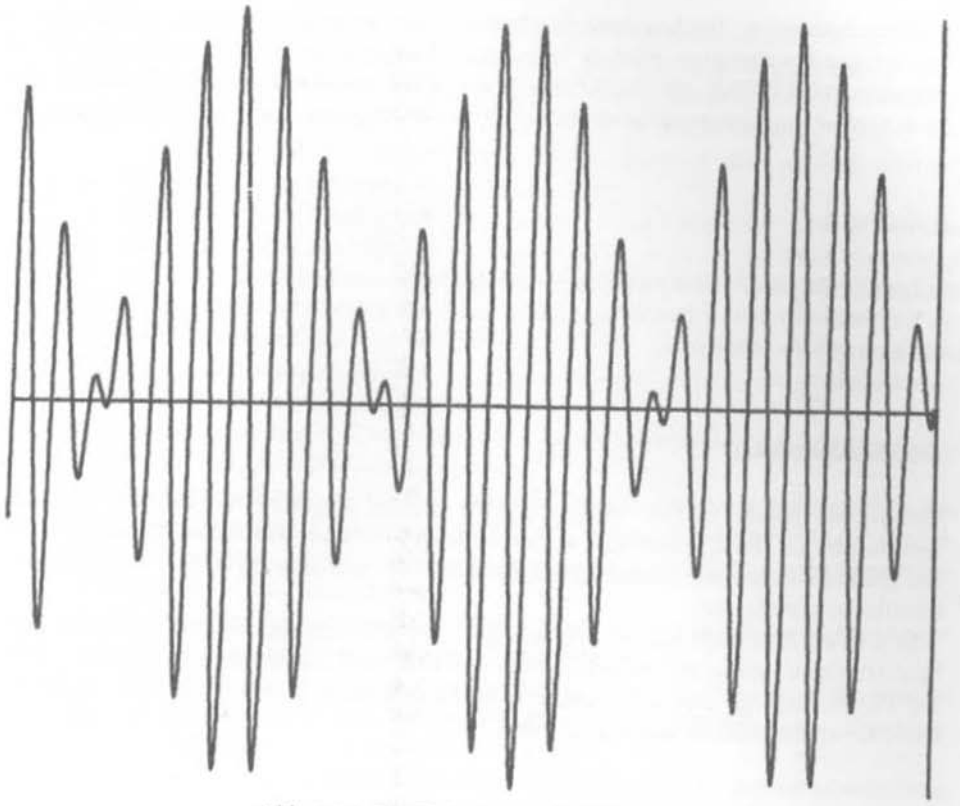


Abb. 3.3.8 Das Ereignis aus dem Plotter

daß sich *Abb. 3.3.8* ergibt. Das Bild ist direkt vom Plotter gezeichnet und fotographisch abgenommen.

Wegen der Länge des Programms sollte der NDR-Klein-Computer mindestens 16KByte RAM besitzen. Dann wird das Programm ab Adresse $A000 mit dem Texteditor eingegeben. Mit der „ORG $9400„-Anweisung am Anfang wird dem Assembler gesagt, daß er den Code auf die Adressen $9400 und folgende legen soll.

Es gibt nun fünf interessante Unterprogramme. Mit „HEBE" wird der Schreibstift angehoben. Mit „SENKE" wird er gesenkt. Mit „MOVETO", das genauso wie das enstprechende Programm im Grundprogramm arbeitet, wird der Plotterstift auf eine neue Position gefahren. Mit „DRAWTO" zeichnet er dorthin eine Linie. Das Unterprogramm „MLINIE" zeichnet gleich eine Linie von einem Anfangspunkt xa;ya nach xe;ye. Man kann den Plotter auch schneller arbeiten lassen, wenn man die Warteschleife bei „WARTE" mit einem kleineren Wert als 2000 versorgt. Wenn man den Wert aber zu klein wählt, arbeiten die Schrittmotoren nicht mehr sauber und führen Schritte nicht aus, die erforderlich wären. Die Folge davon ist, daß dann der Plotter nicht mehr den Weg zurück findet.

Wenn man seine Zeichnungen beschriften will, so muß man einen Zeichensatz in Vektorform konstruieren. Diesen kann man dann mit „MOVETO" und „DRAWTO" ausgeben. ACHTUNG! „MOVETO" und „DRAWTO" werden jetzt ohne „@" aufgerufen, da es eigene Unterprogramme sind und nicht die entsprechenden im Grundprogramm.

Aufgaben:

1. Lassen Sie den Plotter einen Kreis zeichnen.
2. Zeichnen Sie einen Buchstaben „A".
3. Zeichnen Sie ein Gitter.

3.4 Mondlandung

Videospiele haben in den Wohnzimmern Einzug gehalten. Es fing mit einer Art Tennisspiel an, doch die rasante Entwicklung führte bald über Schießspiele bis hin zu der großen Gruppe der Abenteuerspiele. Eine etwas anspruchsvollere Art stellen die Simulationsspiele dar.

In diesem Abschnitt soll gezeigt werden, wie ein einfaches Simulationsspiel aufgebaut ist. Dazu wurde die Mondlandung gewählt, weil sie einige interessante Aspekte der Physik aufzeigt und mit unserer Graphik sehr hübsch zu gestalten ist. Außerdem kann sie später noch ausgebaut werden.

Abb. 3.4.1 Das Pflichtenheft definiert die Aufgabenstellung. Das Struktogramm zum Programm Mondlandung

Nun zu unserer Aufgabe. Eine Mondlandefähre, die auf dem Bildschirm dargestellt wird, soll auf einer Mondoberfläche landen. Dazu muß die Mondlandefähre steuerbar sein. Die Mondlandefähre besitzt hierfür Steuerdüsen in horizontaler und vertikaler Richtung. Sie unterliegt normalerweise der Schwerkraft des Mondes, wird also von diesem angezogen. Wenn man die Landefähre sanft aufsetzen will, so muß man mit Hilfe der Steuerdüsen der Schwerkraft entgegenwirken.

Bevor wir uns an die Programmierung dieser Aufgabe machen, werden wir ein Pflichtenheft (*Abb. 3.4.1*) erstellen. Das Pflichtenheft ist eine wichtige Grundlage komplexer Programmieraufgaben. Dort hinein werden alle Wünsche und Bedingungen geschrieben, die das Programm später erfüllen soll.

Der erste Hauptpunkt besagt, daß eine Mondoberfläche dargestellt werden soll. Dies kann man später zum Beispiel mit der Schildkrötengraphik tun; das Pflichtenheft muß über die Art der Realisierung noch nichts aussagen.

Dann soll ein ebener Landeplatz auf der Mondoberfläche vorhanden sein, denn die Fähre muß ja auf der Mondoberfläche sauber aufsetzen können.

Der dritte und auch der vierte Punkt erklären sich von selbst; natürlich brauchen wir eine bewegliche Landefähre. Diese Landefähre könnte man zwar auch mit der Schildkrötengraphik erzeugen, jedoch werden wir hier eine elegantere Methode kennenlernen, um bewegliche Objekte darzustellen.

Ferner werden Steuerdüsen benötigt, die die Bewegung beeinflussen. Die Anziehungskraft des Mondes muß ebenfalls berücksichtigt werden.

Das Ergebnis eines Landeversuches soll am Schluß des Spiels auf dem Bildschirm angezeigt werden. Dabei soll der Rechner den Text „Gut gelandet" oder „Bruchlandung" auf den Bildschirm schreiben.

Die Angabe bei den Steuerdüsen kann man noch verfeinern. Es soll dabei eine Düse geben, die nach links zeigt, eine soll nach rechts zeigen, eine nach oben und eine nach unten. Diese Düsen muß man getrennt einschalten können. Mit der Düse, die nach unten zeigt, kann man z.B. die Anziehungskraft des Mondes ausgleichen. Vier Tasten des kleinen Zusatztastenfeldes sollen der Steuerung dienen.

Der vorhandene Treibstoff ist zu begrenzen, denn die Mondlandefähre kann nicht unbegrenzt viel Treibstoff mit sich führen.

Hiermit wäre unsere Aufgabe klar formuliert, und wir können uns an die Entwicklung der Lösung machen. Zunächst gilt es, das Vorhaben zu durchdenken, um dann das Ergebnis übersichtlich in einem Struktogramm darzustellen.

Am schwierigsten ist es wohl, die Formeln für die Bewegung der Mondlandefähre zu finden. Zum einen kann sich die Fähre in x-Richtung bewegen, zum anderen in y-Richtung. Man kann diese beiden Bewegungen ungestört einander überlagern, und somit ergeben sich zwei getrennte Formeln. Eine für die x-Richtung und eine für die y-Richtung.

Zunächst gehen wir von der folgenden, bekannten Formel aus:

$x = v \cdot t$

Es wird also ein bestimmter Weg x in einer bestimmten Zeit t mit einer bestimmten Geschwindigkeit v zurückgelegt.

Diese Formel setzt eine konstante Geschwindigkeit voraus. Wenn wir bei unserer Fähre die Steuerdüsen betätigen, so beschleunigen wir aber, ändern also die Geschwin-

digkeit. Darum ist diese Formel für uns nicht unmittelbar zu gebrauchen. Es besteht aber die Möglichkeit, daraus eine brauchbare Berechnung abzuleiten. Wir gehen im Prinzip von derselben Grundüberlegung aus, zerlegen aber unsere Bewegung in so kleine Zeitschritte, daß die Geschwindigkeitsänderung während dieser Zeit vernachlässigbar klein ist. Wir berechnen also jeweils die Strecke, die in einem klein genug gewählten Δt zurückgelegt wurde. Diese Berechnung hat sooft zu erfolgen, bis die Gesamtzeit abgelaufen ist. Δt kann dabei konstant bleiben. Wegen der vorliegenden Beschleunigung ist natürlich bei jeder neuen Berechnung die aktuelle Geschwindigkeit v einzusetzen.

Nun hätten wir zumindest eine Formel für die x-Richtung gefunden. Für die jetzt noch zu lösende Bewegung in der y-Richtung gilt im Prinzip die gleiche Gesetzmäßigkeit; wir müssen lediglich bedenken, daß auf unsere Fähre laufend die Anziehungskraft des Mondes einwirkt. Dieser Tatsache tragen wir dadurch Rechnung, daß wir bei jeder Berechnung einen konstanten Betrag mit negativem Vorzeichen zu unserer Geschwindigkeit in y-Richtung addieren.

Es gibt also jetzt zwei Formeln:

xneu = xalt + vx · Δt

yneu = yalt + vy · Δt

ferner gilt:

vy = vy - Gravitationsanteil

Man konnte die verwendeten Formeln auch exakt ableiten, jedoch ist der mathematische Aufwand dazu recht hoch und auch nicht Lernziel dieses Abschnitts.

Nachdem wir uns jetzt unsere Formeln zurechtgelegt haben, können wir uns den weiteren Überlegungen zum Programm widmen. Sinnvollerweise halten wir dann unsere Gedanken in einem Struktogramm fest.

Die *Abb. 3.4.2* zeigt den ersten Teil. Der erste Schritt im Struktogramm ist die Darstellung der Mondoberfläche. Denn bevor eine Landung durchgeführt werden kann, muß die Mondoberfläche sichbar sein.

Dann werden die Variablen für die Mondlandefähre mit den Anfangswerten vorbelegt. Die x- und y-Position der Fähre wird mit beliebigen Werten a und b belegt. Damit wird festgelegt, wo die Fähre beim Beginn des Spiels erscheinen soll. Δx ist die aktuelle Geschwindigkeit in x-Richtung, also das vx aus der Formel; es wird ebenfalls mit einem Wert vorbelegt. Soll sich die Fähre am Start des Spiels nicht bewegen, so muß dieser Wert mit 0 initialisiert werden. Δy ist ein Maß für die Geschwindigeit in y-Richtung, also für das vy aus unserer Formel. Der Treibstoffvorrat wird schließlich mit einem Wert c belegt. Je kleiner man c wählt, desto weniger Treibstoff steht dem Spieler zur Lösung der Aufgabe zur Verfügung.

Der innere Block, den wir mal Spielblock nennen, soll dann solange ausgeführt werden, bis die Fähre eine y-Koordinate hat, die kleiner oder gleich 0 ist. Dies bedeutet also, bis die Fähre die Mondoberfläche berührt. Danach muß festgestellt werden, ob es sich um eine normale Landung handelte, oder ob Bruch gemacht wurde. Dazu kann man zum Beispiel die aktuelle Geschwindigkeit in x- und y-Richtung heranziehen, also die Variablen Δx und Δy.

Wenn der Betrag von Δx einen Wert d unterschreitet und der Betrag von Δy einen Wert e, so soll eine erfolgreiche Landung durchgeführt worden sein. Die Werte d und e

bestimmen dann, welcher Bereich bei einer Landung zugelassen wird. Je kleiner man e und d wählt, desto schwieriger wird eine Landung, umso sanfter muß sie für ein Bestehen der Prüfung ausgeführt werden. Ist Δx sehr groß, so besitzt die Fähre eine große Geschwindigkeit in x-Richtung, und sie würde bei einer realen Landung wohl Purzelbäume schlagen. Man muß den Betrag abfragen, denn die Fähre kann sowohl eine positive, als auch eine negative Geschwindigkeit in der x-Richtung haben. Bei einer negativen Geschwindigkeit bewegt sich die Fähre von links nach rechts und bei einer positiven eben umgekehrt. Entsprechendes gilt für Δy.

Nun zum inneren Block. Die *Abb. 3.4.3* zeigt das Struktogramm, also die Verfeinerung unseres ersten Struktogramms aus der Abb. 3.4.2. Als erstes wird die Fähre bei der Position x;y ausgegeben. Dann ist festzustellen, ob noch Treibstoff vorhanden ist. Ist dies nicht der Fall, so fällt die Fähre manövrierunfähig im freien Fall, also noch der Anziehungskraft des Mondes ausgesetzt, auf die Mondoberfläche. Wenn noch Treibstoff vorhanden ist, so soll sich die Eingabe per Tastenfeld auf die Landefähre auswirken, es ist also die Tastatur abzufragen.

Es sollen vier Steuertasten verwendet werden. Wenn man die Taste „Düse rechts" drückt, so wird Δx um einen Wert erhöht. Hier wurde z. B. der Wert 3 gewählt. Die Größe des Wertes gibt praktisch die Schubkraft der Düse an. Wenn man die Taste „Düse links" betätigt, so wird Δx um den Wert 3 verringert. Bei der Taste „Rauf", wird Δy um den Wert 3 erhöht. Damit kann man die Fähre weg von der Oberfläche beschleunigen, und bei „Runter" wird Δy um 3 verringert. Dadurch kann man schneller auf die Mondoberfläche gelangen.

Nach dem Abfrage-Block wird Δy um eins verringert. Dadurch wird die Anziehungskraft des Mondes realisiert. Der Wert, um den man Δy verringert, muß kleiner sein, als der Wert bei der entsprechenden Steuerdüse, sonst kann man die Fähre nicht mehr bremsen. An dieser Stelle ist auch die Verzweigung, abhängig von der Abfrage nach dem Treibstoff, wieder beendet, denn die Anziehungskraft soll sich ja in jedem Fall auf unser Gefährt auswirken. Genaugenommen dürfte hier für die Gravitation natürlich keine Konstante verwendet werden. Sie ändert sich in Abhängigkeit von der Flughöhe. Wir nehmen zur Vereinfachung an, die Fähre befinde sich bereits so knapp über der Oberfläche, daß sich dieser Umstand nicht mehr deutlich auswirkt.

Danach erfolgt die Berechnung der neuen Position. Dazu wird auf die alte x-Koordinate das neue Δx addiert und auf die y-Koordinate das neue Δy. Anschließend wird der Treibstoffvorrat um eins verringert.

Schließlich folgt noch eine Warteschleife zur Definition der Zeit Δt, die ja ebenfalls wesentlicher Bestandteil unserer Berechnung war. Würden wir hier die durch den Computer erreichbare Arbeitsgeschwindigkeit verwenden, so wäre die Darstellung der Landung auf dem Bildschirm nicht realistisch.

Nun haben wir noch ein kleines Problem, nämlich die Darstellung der Landefähre. Zur Realisierung der Fähre gibt es im Grundprogramm ein nützliches Unterprogramm. Dieses Unterprogramm eignet sich speziell für bewegte Objekte und heißt „FIGUR". Ein bewegtes Objekt wird dazu in einzelne Vektoren zerlegt, die man dem Unterprogramm in codierter Form zur Verfügung stellt. Der Figur-Befehl löscht dann immer erst die alte Figur, falls eine solche vorhanden war, und setzt anschließend die neue Figur auf den

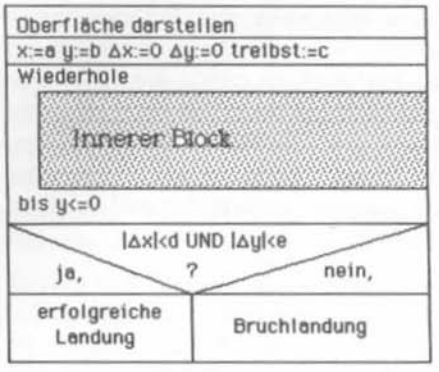


Abb. 3.4.2 Struktogramm: Mondlandung

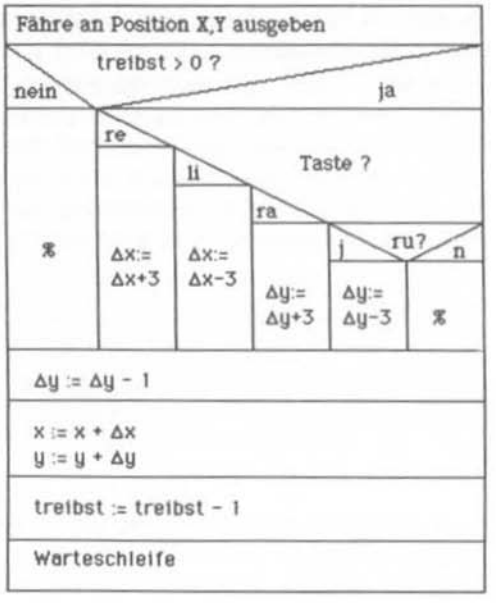


Abb. 3.4.3 Der innere Block

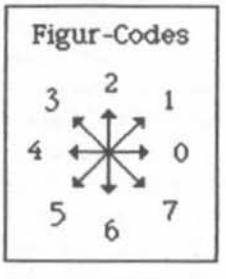


Abb. 3.4.4 Die Codes des Figur-Befehls

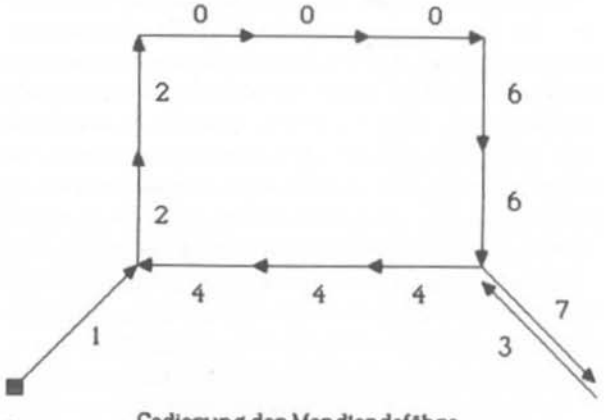


Abb. 3.4.5 Die Codierung der Mondlandefähre

0,1,2,3,4,5,6,7	Richtungscode
8	Schreibstift heben
9	Schreibstift senken
10	Ende der Tabelle

Tabelle 3.4.1 Zusammenfassung der Figur-Codes

Schirm. Als Parameter wird dem Unterprogramm dazu noch die neue x- und y-Koordinate übergeben.

Die *Abb. 3.4.4* zeigt die Codierung der Vektoren. Die Vektoren können dabei nur einen Winkel von 45° unterscheiden. In der *Abb. 3.4.5* ist eine Landefähre, in einzelne Vektoren zerlegt, dargestellt. Wie wir es schon von unserer Schildkröte her kennen, so gibt es auch hier die Möglichkeit, den Schreibstift zu heben, also unsichtbar zu zeichnen; dafür ist der Code 8 vorgesehen. Mit 9 wird er dann wieder gesenkt. Außerdem ist noch der Code 10 als Endekennung der Codetabelle definiert. In der *Tabelle 3.4.1* sind die Codes nochmals in einer Übersicht dargestellt.

Die Pseudobefehle dc.b, dc.w, dc.l und ds

Abb. 3.4.6 zeigt ein kleines Testprogramm, das die Landefähre auf den Bildschirm bringt. Im Vereinbarungsbaustein werden zunächst die Vektoren in einer Tabelle definiert. Dazu erhält die Tabelle den Namen „faehre". Dann folgt ein Befehl an den Assembler: „dc.b" bedeutet „define constant byte". Also definiere Byte-Konstante. Wenn man anstelle von ".b" ein ".w" oder ".l" schreibt, so kann man auch Worte oder Langworte (32 Bit) definieren. Nach diesem sogenannten Pseudobefehl folgt eine Liste von Werten. Die Zahlenwerte werden direkt im Speicher abgelegt. Also „dc.b 1" legt den Wert 01 im Speicher als ein Byte ab. Mehrere solcher Werte darf man, durch Komma getrennt, in eine Zeile schreiben. Nach der Tabelle folgt der Befehl „ds 0". Auch das ist eine Anweisung an den Assembler. „ds" bedeutet „define storage", also definiere einen Speicherbereich. Damit kann man die Anzahl Speicherzellen frei halten, die hinter ds als Zahl angegeben werden, ohne sie mit einem Wert zu belegen. Wenn man den Wert 0 als Dimension angibt, so besitzt der Befehl eine besondere Bedeutung. Er reserviert genau dann ein Byte Speicherplatz, wenn die nächste freie Speicherzelle auf einer ungeraden Adresse zu liegen kommt. Dieser Befehl wird gebraucht, weil beim 68000/8 Befehle und Worte, wie auch Langworte immer nur bei geraden Adressen beginnen dürfen. Ist dies nicht der Fall, so erfolgt die Fehlermeldung „ADRESS ERROR". Diese Einschränkung kommt daher, daß der 68000 Befehle immer als ein Wort, also mit 16 Bit holt. Beim 68008 hat man diese Einschränkung beibehalten, um kompatibel zu bleiben. Nachdem wir uns das Abzählen unserer Tabelle sparen wollen, es sich aber um eine Tabelle mit Byte-Einträgen handelt, setzen wir den ds 0-Befehl ein.

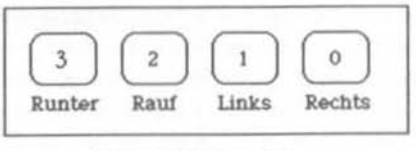
Das Programm beginnt dann bei der Marke „start". Als erstes wird die Adresse der Tabelle „faehre" in das Adreßregister A0 geladen. Der Befehl „LEA" ist die Abkürzung für „load effective adress", also „lade die Adresse". Als Ziel wird das Register A0 angegeben. Anstelle des „LEA"-Befehls hätte man auch den Befehl „MOVEA.L #FAEHRE,A0" verwenden können. Der „LEA"-Befehl ist eleganter und weist automatisch auf ein Adreßregister hin.

Danach wird der Wert 3 in das Register D0 geladen. Im Register D0 wird der Vergrößerungsfaktor für den Figur-Befehl angegeben. Wenn man den Wert 1 in D0 angibt, so ist ein Vektor genau einen Bildschirmpunkt lang. Will man größere Vektoren darstellen, so kann man mehrere Vektoren aneinanderreihen, oder, wie in unserem Fall,

```
* Vereinbarungsbaustein

faehre:
 dc.b 1,2,2,0,0,0,6,6,7,3,4,4,4,10
 ds 0

* Programmbaustein

start:
 lea faehre,a0   * Adresse der Faehre
 move #3,d0      * Groesse der Faehre
 move #100,d1    * X-Koordinate
 move #180,d2    * Y-Koordinate
 jsr @figur      * und ausfuehren
 rts

 end
```

Abb. 3.4.6
Ein erster Test

```
Textstart=009000 Fenster=009000 Tor=009000      aner CTRL-U=Hilfe
```



Anschluß der Tasten

Abb. 3.4.7 Die Belegung der Tasten

einen größeren Faktor übergeben. Der Wert ist dabei als Länge in Punkten definiert. Zu beachten ist lediglich noch, daß die Werte bis einschließlich 3 unmittelbar vom „FIGUR"-Unterprogramm an den Graphikprozessor übergeben werden und somit sehr schnell in der Ausführung sind. Werte darüber müssen vom Grundprogramm erst in einer Schleife umgesetzt werden und bewirken somit eine langsamere Ausführung. Wird der Wert 0 als Vergrößerungsfaktor übergeben, so wird nur die alte Figur gelöscht, aber keine neue gezeichnet.

Im Register D1 erwartet der FIGUR-Befehl die x-Koordinate, hier 100, und im Register D2 die y-Koordinate. Dabei kann man für die x-Koordinate einen Bereich von 0 bis 511 angeben und für die y-Koordinate einen Bereich von 0 bis 255, wenn die Figur im sichtbaren Bildfenster liegen soll. Prinzipiell ist der Bereich 0 bis 4095 für jede Koordinate erlaubt, bei größeren Werten erscheint die Figur wieder auf dem Bildschirm.

Damit wären die Vorbereitungen für den Befehl erledigt, wir können also die Abarbeitung durch den Aufruf „JSR @FIGUR" starten.

Wenn man anschließend die Figur auf eine andere Position setzen würde, so würde die alte Figur automatisch gelöscht. Dieses Löschen geht so schnell, daß sich dadurch ein Bewegungseindruck ergibt. Es wird auch dann automatisch gelöscht, wenn die neue Figur mit der alten nicht identisch ist, oder einen anderen Vergrößerungsfaktor aufweist. Das Grundprogramm merkt sich nicht nur die alte Position der Figur, sondern auch die Adresse der Tabelle. Darum sollte man niemals die Tabelle unmittelbar verändern, sonst würde das Grundprogramm beim nächsten Aufruf der „FIGUR"-Funktion eine Figur löschen, die gar nicht vorhanden ist. Bevor wir nun die Besprechnung dieses Befehls beenden, sei der Vollständigkeit halber noch auf eine weitere Besonderheit hingewiesen. So kann man mit dem Unterprogramm „SETFIG" eine Figur auf dem Bildschirm einfrieren. Bei einem erneuten Aufruf von „FIGUR" bleibt die alte Figur auf dem Bildschirm stehen, und es wird nur die neue Figur gezeichnet.

Das Tastenfeld des Abschnitts „Schrittmotor steuern" verwenden wir auch hier wieder für die Eingaben. Die IOE-Baugruppe wird mit diesem Tastenfeld verbunden. Dabei wird die IOE-Baugruppe, wie gehabt, auf die Adresse $30 eingestellt. *Abb. 3.4.7* zeigt die Belegung des Tastenfeldes. Es werden nur die rechten vier Tasten verwendet. Die Taste vom Bit 0 der IOE-Baugruppe ist die Taste für „Düse rechts", die nächste Taste steuert die linke Düse, usw.

Abb. 3.4.8 zeigt das komplette Listing des Mondlandeprogramms. Dabei tauchen eine Reihe von neuen Befehlen auf, die wir nun nacheinander besprechen werden. Im Vereinbarungsbaustein wird zunächst die Portadresse des Tastenfeldes durch EQU definiert. Dann folgen die sogenannten Bitmaskendefinitionen der einzelnen Tasten. Bei der späteren Tastenabfrage ist somit eine einfache und übersichtliche Programmierung möglich.

```
Rolf-D.Klein 68000/08 Assembler 4.3 (C) 1984, Seite 1


009C00                  ;*********************************
009C00                  ; MONDLANDEPROGRAMM VERSION 1.0 ;
009C00                  ; VOM 13.12.1983                ;
009C00                  ;*********************************
009C00
00A000                   ORG $A000
00A000
00A000                  ; ***** VEREINBARUNGS-BAUSTEIN *****
00A000
= FFFFFF30              TASTEN EQU $FFFFFF30
= 00000001              RECHTS EQU 1    ; BIT MASKEN
= 00000002              LINKS  EQU 2
= 00000004              RAUF   EQU 4
= 00000008              RUNTER EQU 8
00A000
00A000                  ERFOLGTEXT:
00A000  4775742067656C   DC.B 'Gut gelandet',0
00A007  616E64657400
00A00D
00A00D                  BRUCHTEXT:          zu Abb. 3.4.8
```

```
Rolf-D.Klein 68000/08 Assembler 4.3 (C) 1984, Seite 2


00A00D  42727563686C61    DC.B 'Bruchlandung',0
00A014  6E64756E6700
00A01A
00A01A                    FAEHRE:
00A01A  08 02 09 00 00    DC.B 8,2,9,0,0,7,3,2,2,4,4,4,4
00A01F  07 03 02 02 04
00A024  04 04 04
00A027  06 06 05 01 00    DC.B 6,6,5,1,0,0,10
00A02C  00 0A
00A02E
00A02E                     DS 0 ; ANGLEICH AUF WORTGRENZE
00A02E
00A02E                    ; VARIABLENVEREINBARUNGEN
00A02E
00A02E  0000              X: DC.W  0
00A030  0000              Y: DC.W  0
00A032  0000              DX: DC.W  0
00A034  0000              DY: DC.W  0
00A036  0000              TREIBST: DC.W  0
00A038
00A038                    ; ***** UNTERPROGRAMM-BAUSTEIN *****
00A038
00A038                    MONDFLAECHE:
00A038  323C 0000          MOVE #0,D1
00A03C  343C 0000          MOVE #0,D2
00A040  363C 005A          MOVE #90,D3
00A044  4EB9 0000131A      JSR @SET        ;SETZE SCHILDKROETE
00A04A  4EB9 00001112      JSR @HIDE       ;ABER NICHT ZEIGEN
00A050  383C 0031          MOVE #50-1,D4     ;ANZAHL KRATER
00A054                    MLP1:
00A054  4EB9 0000A060      JSR KRATER
00A05A  51CC FFF8          DBRA D4,MLP1
00A05E  4E75               RTS
00A060
00A060                    KRATER:
00A060  303C FFD3          MOVE  #-45,D0
00A064  4EB9 000012F6      JSR @DREHE
00A06A  303C 000A          MOVE #10,D0
00A06E  4EB9 00001288      JSR @SCHREITE
00A074  303C FFD3          MOVE #-45,D0
00A078  4EB9 000012F6      JSR @DREHE
00A07E  303C 0006          MOVE #6,D0
00A082  4EB9 00001288      JSR @SCHREITE
00A088  303C FFD3          MOVE #-45,D0
00A08C  4EB9 000012F6      JSR @DREHE
00A092  303C 000A          MOVE #10,D0
00A096  4EB9 00001288      JSR @SCHREITE
00A09C  303C FF1F          MOVE #-45-180,D0
00A0A0  4EB9 000012F6      JSR @DREHE
00A0A6  4E75               RTS
00A0A8
00A0A8                    ; ***** PROGRAMM-BAUSTEIN *****
```

zu Abb. 3.4.8

```
00A0AB
00A0A8                    START:
00A0AB  6100 FF8E          BSR MONDFLAECHE
00A0AC  33FC 0064          MOVE #100,X
00A0B0  0000A02E
00A0B4  33FC 00C8          MOVE #200,Y
00A0B8  0000A030
00A0BC  33FC 0003          MOVE #3,DX
00A0C0  0000A032
00A0C4  33FC 0000          MOVE #0,DY
00A0C8  0000A034
00A0CC  33FC 012C          MOVE #300,TREIBST
00A0D0  0000A036
00A0D4                    WDH1:
00A0D4  207C 0000A01A      MOVEA.L #FAEHRE,A0
00A0DA  303C 0003          MOVE #$3,D0              ; VERGROESSERUNG=3
00A0DE  3239 0000A02E      MOVE X,D1
00A0E4  3439 0000A030      MOVE Y,D2
00A0EA  4EB9 0000328C      JSR @FIGUR
00A0F0  4A79 0000A036      TST TREIBST
00A0F6  6F00 0052          BLE NOTEST               ; KEIN TREIBSTOFF ?
00A0FA  1039 FFFFFF30      MOVE.B TASTEN,D0
00A100  0200 0001          AND.B #RECHTS,D0
00A104  6608               BNE.S TST1
00A106  5679 0000A032      ADDQ #3,DX
00A10C  603C               BRA.S NOTEST
00A10E
00A10E                    TST1:
00A10E  1039 FFFFFF30      MOVE.B TASTEN,D0
00A114  0200 0002          AND.B #LINKS,D0
00A118  6600 000A          BNE TST2
00A11C  5779 0000A032      SUBQ #3,DX
00A122  6026               BRA.S NOTEST
00A124                                              zu Abb. 3.4.8
00A124                    TST2:
00A124  1039 FFFFFF30      MOVE.B TASTEN,D0
00A12A  0200 0004          AND.B #RAUF,D0
00A12E  6608               BNE.S TST3
00A130  5679 0000A034      ADDQ #3,DY
00A136  6012               BRA.S NOTEST
00A138
00A138                    TST3:
00A138  1039 FFFFFF30      MOVE.B TASTEN,D0
00A13E  0200 0008          AND.B #RUNTER,D0
00A142  6606               BNE.S NOTEST
00A144  5779 0000A034      SUBQ #3,DY
00A14A
00A14A                    NOTEST:
00A14A  5379 0000A034      SUBQ #1,DY               ;GRAVITATION
00A150  3639 0000A032      MOVE DX,D3               ; X:=X+DX
00A156  D779 0000A02E      ADD D3,X
00A15C  3639 0000A034      MOVE DY,D3               ; Y:=Y+DY
00A162  D779 0000A030      ADD D3,Y
00A168  5379 0000A036      SUBQ #1,TREIBST          ; TREIBSTOFF VERINGERN
00A16E
```

```
00A16E  323C 0004        MOVE #5-1,D1          ; WARTESCHLEIFE
00A172                  WARTE20MS:
00A172  3E3C 001C        MOVE #!SYNC,D7
00A176  4E41             TRAP #1
00A178  6700 FFF8        BEQ WARTE20MS
00A17C  51C9 FFF4        DBRA D1,WARTE20MS
00A180
00A180  0C79 0005        CMP #5,Y              ; HOEHE DER OBERFLAECHE UEBEF
00A184  0000A030
00A188  6E00 FF4A        BGT WDH1              ; BERUECKSICHTIGEN
00A18C                                         ;
00A18C  303C 0003        MOVE #$3,D0           ; FAEHRE AM SCHLUSS NOCHMALS
00A190  3239 0000A02E    MOVE X,D1
00A196  3439 0000A030    MOVE Y,D2
00A19C  4EB9 0000328C    JSR @FIGUR            ; FIGUR AUFRUFEN
00A1A2  3039 0000A032    MOVE DX,D0
00A1A8  6A02             BPL.S FF1
00A1AA  4440             NEG D0
00A1AC                  FF1:                   ; !DX! < 2
00A1AC  0C40 0002        CMP #2,D0
00A1B0  6C18             BGE.S BRUCH
00A1B2  3039 0000A034    MOVE DY,D0
00A1B8  6A02             BPL.S FF2
00A1BA  4440             NEG D0
00A1BC                  FF2:                   ; !DY! < 4
00A1BC  0C40 0004        CMP #4,D0
00A1C0  6C08             BGE.S BRUCH
00A1C2  41F9 0000A000    LEA ERFOLGTEXT,A0
00A1C8  6006             BRA.S ENDE
00A1CA
00A1CA                  BRUCH:
00A1CA  41F9 0000A00D    LEA BRUCHTEXT,A0
00A1D0                  ENDE:
00A1D0  303C 0033        MOVE #$33,D0          ; SCHRIFTGROESSE
00A1D4  323C 0000        MOVE #0,D1            ; X-POSITION
00A1D8  343C 0082        MOVE #130,D2          ; Y-POSITION
00A1DC  4EB9 00001386    JSR @WRITE
00A1E2  4EB9 00000A00    JSR @CI               ; WARTEN AUF TASTE
00A1E8  4EB9 00000DA0    JSR @CLR              ; LOESCHEN
00A1EE  6000 FEB8        BRA START             ; UNBEGRENZT
00A1F2
00A1F2                  ; ***********
00A1F2
00A1F2                  END

008F4C  Ende-Symboltabelle
```

Abb. 3.4.8 Das Mondlande-Programm

Mit dem „dc.b"-Befehl kann man nicht nur Zahlenwerte im Speicher ablegen, wie wir es bisher bereits kennengelernt haben, sondern auch Texte. Dazu wird der Text zwischen Anführungszeichen geschrieben. So werden hier die Texte „Gut gelandet" und „Bruchlandung" im Speicher als ASCII-Text abgelegt. Mit ",0" nach jedem Text wird der Code 00 im Speicher abgelegt. Das Grundprogramm benötigt diesen Wert als Erkennung für das Textende.

Die dann folgende Tabelle zur Definition der Figur „FAEHRE" und der Befehl „ds 0" sind uns bereits bekannt. Danach folgen die Variablenvereinbarungen. Dort stehen Speicherzellen, die im Verlauf des Programms verändert werden, also als Variablen Verwendung finden. Zum Beispiel die Variable für die x-Koordinate. Sie ist mit „X:" markiert. Mit „dc.w 0" wird ein 16-Bit-Wort reserviert und der Wert 0000 als Startwert abgelegt. Dann folgen die Variablen Y für die y-Koordinate, DX für den Wert Δx und DY für den Wert Δy, sowie TREIBST, worin der aktuelle Treibstoffvorrat gespeichert wird.

Weiter geht es mit den Unterprogrammbausteinen. Im Programm „MONDFLAECHE" wird die Mondoberfläche mit Hilfe der Schildkrötengraphik erzeugt. Im vorliegenden Fall des Mondlandeprogramms soll hier die Oberfläche nur mit kleinen Kratern übersät sein. Man kann dann überall landen. Dazu wird das Unterprogramm „KRATER" 50 mal aufgerufen. Mit dem Befehl „SET" wird die Startposition der Schildkröte festgelegt. Die Register D1 und D2, die für die x- und y-Koordinate stehen, werden mit 0 besetzt und das Register D3 mit dem Wert 90. Im Register D3 wird die Blickrichtung der Schildkröte festgesetzt. Damit befindet sich die Schildkröte ganz links unten und blickt senkrecht nach oben. Durch Aufruf des Unterprogramms „HIDE" wird verhindert, daß die Schildkröte selbst im Bild erscheint, denn auf dem Mond gibt es keine Schildkröten. Mit dem „DBRA"-Befehl wird die Schleife realisiert, die 50 mal das Unterprogramm „KRATER" aufruft. Im Unterprogramm „KRATER" wird mit Hilfe von „DREHE" und „SCHREITE" der Krater gezeichnet.

Dann folgt, beginnend bei der Marke „START", das Hauptprogramm. Mit dem Befehl „BSR MONDFLAECHE" wird dort zuerst, wie im Struktogramm vorgesehen, die Mondoberfläche ausgegeben. Hier wurde zur Abwechslung mal der Befehl „BSR" anstelle von „JSR" verwendet. Der „BSR"-Befehl benötigt weniger Speicherplatz, da er relativ adressiert. Es muß also im Maschinencode nur die Differenz zwischen der aktuellen Adresse und dem Sprungziel angegeben werden, während der „JSR"-Befehl absolut adressiert, also die Zieladresse direkt enthält. Ein weiterer Unterschied ist die Abarbeitung im Einzelschritt. Das Grundprogramm führt den „JSR"-Befehl als Ganzes aus, das Unterprogramm wird also als ein Schritt betrachtet und komplett abgearbeitet. Bei der Verwendung des „BSR"-Befehls werden auch die Befehle der aufgerufenen Unterroutine schrittweise abgearbeitet. Nachdem ein möglicherweise vorhandener Fehler auch im Unterprogramm liegen kann, bietet uns diese Einrichtung bei der Fehlersuche einen klaren Vorteil. Der „BSR"-Befehl hat noch einen weiteren, wichtigen Vorteil. Soll ein Programm an verschiedenen Stellen des Speichers lauffähig sein, so stellt der BSR-Aufruf eine wesentliche Voraussetzung dazu dar. Die relative Distanz verändert sich beim Verschieben ja nicht. Natürlich müssen dann auch alle anderen Sprünge und auch die Bedienung des Variablenteils diesem Grundsatz folgen, um ein Programm wirklich verschiebbar zu gestalten. Welche Dinge hier im Detail zu beachten sind, werden wir in einem späteren Kapitel dieses Buches noch erfahren.

Im Struktogramm steht als nächstes die Initialisierung der Variablen. Die Fähre soll bei einer Position x=100 und y=200 starten. Mit dem „MOVE"-Befehl werden die Anfangswerte dazu in die Variablen X und Y transportiert. Um der Fähre eine Anfangsgeschwindigkeit in x-Richtung zu geben, belegen wir die Variable DX mit 3. Man muß die Fähre also auch in x-Richtung beeinflussen, um eine Landung durchzuführen. Die

Variable DY wird mit dem Wert 0 belegt, somit hat die Fähre also im ersten Moment noch keine Geschwindigkeit in y-Richtung. An Treibstoff sehen wir 300 Einheiten vor.

Dann folgt die Marke für die Wiederholschleife „WDH1". Dort beginnt der Spielblock, der im ersten Struktogramm als „Innerer Block" bezeichnet wurde. Zunächst wird die Fähre an der Position x;y ausgegeben. Mit „MOVE X,D1" wird die x-Koordinate in das Register D1 übertragen. „MOVE Y,D2" transportiert die y-Koordinate in das Register D2, und davor wird mit „MOVE #3,D0" die Größe der Landefähre festgelegt. Damit kann nun der „FIGUR"-Befehl aufgerufen werden, der die Fähre auf dem Bildschirm erscheinen läßt.

Die Assemblerbefehle TST, ADDQ, SUBQ und MOVEQ

Jetzt lernen wir einen neuen Befehl kennen: „TST TREIBST". Damit werden die Bedingungsflags des 68000/8 gesetzt. Es wird geprüft, ob der Inhalt der Variablen TREIBST null ist; wenn ja, so wird das Z-Flag (zero, null) gesetzt. Ebenfalls wird geprüft, ob der Wert kleiner als null ist und wenn ja, so wird das M-Flag (Minus) gesetzt. All dies macht der „TST"-Befehl automatisch, ohne Register zu verändern. Danach kann man z. B. mit „BLE" (branch less or equal) die Entscheidung „TREIBST = 0" durchführen. Wenn der Inhalt der Variablen TREIBST kleiner oder gleich 0 war, so erfolgt ein Sprung zur Marke „NOTEST". In unserem Struktogramm war dieser Vorgang, also das Überspringen der Tastaturabfrage, durch das Zeichen % gekennzeichnet. Im anderen Fall, also wenn TREIBST größer als null ist, läuft das Programm weiter und gelangt zum Befehl „MOVE.B TASTEN,D0". Damit wird der Inhalt des Tastenfeld-Ports in das Register D0 eingelesen. Ist zu diesem Zeitpunkt eine Taste gedrückt, so ist das entsprechende Bit auf den Wert 0 gesetzt. Um unserem Struktogramm gerecht zu werden, ist es jetzt erforderlich, abhängig von der gedrückten Taste, also von einem bestimmten 0-Bit im Register D0, unsere DX- oder DY-Variable zu verändern. Darum wird eine Und-Verknüpfung durchgeführt. Dabei wird der Wert der Konstanten RECHTS mit D0 verknüpft. RECHTS hat den Wert 1, und so ist nach der Verknüpfung der Inhalt von D0.B gleich 0, wenn das Bit 0 vor der Verknüpfung 0 war, also die Taste „RECHTS" gedrückt wurde, oder ungleich 0, wenn die Taste nicht gedrückt war. Das Verknüpfungsergebnis hängt nur von der Lage der Taste „Düse rechts" ab und nicht von den restlichen Tastenstellungen. Mit dem Sprungbefehl „BNE TST1" springt das Programm zur Marke „TST1", wenn die Taste „Düse rechts" nicht betätigt wurde. War die Taste gedrückt, so wird hingegen der Wert 3 mit dem Befehl „ADDQ #3,DX" auf die Variable DX addiert. Anschließend wird ein Sprung zur Marke NOTEST ausgeführt. Im Struktogramm ist das der erste gemeinsame Kasten nach der tastenabhängigen Verzweigung. Das „Q" hinter dem „ADD" ist die Abkürzung für „QUICK" und bedeutet, daß der Befehl einen besonders kompakten Code erzeugt. Man kann diesen Befehl immer dann verwenden, wenn man kleine Konstanten addieren will. Der Bereich ist auf 1 bis 8 begrenzt. Man hätte aber hier genausogut „ADD #3,DX" schreiben können. Der 68000/8 besitzt drei solcher „Quick"-Befehle, den „ADDQ", „SUBQ" und „MOVEQ". Die ersten beiden Befehle kann man für den Zahlenbereich 1 bis 8 verwenden, den letzten Befehl,

der nur für eingebaute Register (D0..D7,A0..A7) funktioniert, für den Zahlenbereich -128 bis +127. Die Befehle sind bei der Ausführung schneller als der entsprechende Befehl ohne dem Q am Ende, außerdem benötigen sie weniger Speicherplatz.

Bei der Marke „TST1" erfolgt nun entsprechend die Abfrage für die Taste „Düse links"; auch diese Abfrage wird durch eine entsprechende Und-Verknüpfung realisiert. Die Taste „Düse links" ist am Bit 1, also am Bit mit der Wertigkeit 2 angeschlossen. Die zur Und-Verknüpfung herangezogene Konstante LINKS enthält ebenfalls den Wert 2. War hier die Taste gedrückt, so liefert uns auch hier die Verknüpfung den Wert 0 im Register D0.B. Auch die weiter unten notwendigen Abfragen für die beiden übrigen Tasten werden so realisiert, die Konstanten dafür haben wir ja bereits definiert. Nachdem die Tasten abgefragt wurden, folgt nun die Marke „NOTEST". Dort wird der Rest des inneren Blocks durchgearbeitet. Der Inhalt der Variablen DY wird verringert, und Koordinaten sind durch die jeweiligen Delta-Werte auf den aktuellen Stand zu versetzen. Außerdem ist der Treibstoffvorrat zu reduzieren, so wie wir es in unserem Struktogramm geplant hatten.

Man könnte das Programm auch so abändern, daß der Treibstoffvorrat nur dann verändert wird, wenn die Düsen in Betrieb sind. Allgemein sind für Veränderungen des Spiels der Phantasie keine Grenzen gesetzt.

Nun folgt die Marke „WARTE20MS". Dort wird das Unterpogramm „SYNC" aufgerufen, das sich im Grundprogramm befindet. Hier mal zur Abwechslung mit dem „TRAP"-Befehl, doch man hätte genausogut einen „JSR"-Befehl verwenden können. Der Befehl „SYNC" liefert als Ergebnis nur ein Flag. Das Zero-Flag (Null-Flag) ist genau dann gesetzt, wenn kein Synchronsignal am Ausgang des Graphikprozessors vorhanden ist. Dieses Synchronsignal ist das vertikale Synchronsignal, das alle 20 Millisekunden erscheint. Ist das Ergebnis nach dem SYNC-Aufruf ungleich null, so ist der Puls aufgetreten. Das Unterprogramm verhindert übrigens auch, daß nach erneutem Aufruf das Flag immer noch gelöscht ist, also ein Wert ungleich null erscheint, bevor nicht wirklich ein neuer Synchronpuls erschienen ist. Die Schleife wird insgesamt 5 mal durchlaufen, um so eine gesamte Verzögerung von 100 Millisekunden zu erreichen. Damit wird verhindert, daß der Bewegungsablauf zu schnell wird. Nun wird die Höhe der Fähre mit dem Wert 5 verglichen. Wenn die Fähre oberhalb von 5 liegt, so wird die Schleife „WDH1" wiederholt, sonst nicht. Hier ergibt sich ein Unterschied zum Struktogramm. Dort haben wir den Wert 0 als Mondoberfläche definiert. Die Mondoberfläche liegt aber hier nicht auf der y-Koordinate 0, sondern oberhalb, daher muß die Abfrage entsprechend geändert werden.

Die Assemblerbefehle CMP und BGT

Mit dem „CMP #5"-Befehl wird der Vergleich durchgeführt. Dabei werden auch hier nur die Bedingungsflags gesetzt. Wenn die y-Koordinate größer als 5 war, so löst der BGT-Befehl einen Sprung zur Marke WDH1 aus, die Schleife wird also wiederholt.

Sonst wird als nächstes die Landefähre in der neuen Position ausgegeben. Zur Entscheidung, ob die Landung geglückt ist oder nicht, werden danach die Werte DX

und DY abgefragt. Die Betragsbildung erfolgt mit dem Befehl „NEG D0", der nur dann aufgerufen wird, wenn der betreffende Wert kleiner als Null war. Dann erfolgt mit „CMP #2" die Abfrage, ob der Betrag von DX kleiner als 2 war, und mit „CMP #4", ob der Betrag von DY kleiner als 4 war.

Der Grundprogrammbefehl WRITE

Die Ausgabe des Textes „Gut gelandet" oder „Bruchlandung" erfolgt mit dem Unterprogramm „WRITE", das ebenfalls im Grundprogramm vorhanden ist. Das Unterprogramm „WRITE" erhält dazu im Register A0 die Adresse des ASCII-Textes im Speicher, der ausgegeben werden soll. Als Endekennung muß die Zeichenfolge mit dem Code 00 abgeschlossen sein. Im Register D1 muß die x-Koordinate, die hier mit 0 angegeben wird, und im Register D2 die y-Koordinate stehen. Auch für die Ausgabe der Zeichen kann ein Vergrößerungsfaktor, sogar für beide Koordinaten getrennt, übergeben werden. Er wird im Register D0 erwartet, die oberen 4 Bit stehen dabei für die y-, die unteren 4 Bit für die x-Koordinate. Der Code berechnet sich dabei nach folgender Formel:

Wert := Textbreite × 16 + Texthöhe

Wenn man den Wert 0 einsetzt, so wird der Vergrößerungsfaktor 16 gewählt.

Je nachdem, ob das Programm zur Marke „BRUCH" springt oder über FF2 und den Sprung an die Marke „ENDE" gelangt, steht im Register A0 die Adresse des Textes „Bruchlandung" oder die Adresse des Textes „Gut gelandet".

Nach dem „WRITE"-Befehl folgt der Aufruf des Unterprogramms „CI". Das Unterprogramm wartet auf die Betätigung einer Taste. Wenn man die Taste drückt, so kehrt das Unterprogramm mit dem ASCII-Wert im Register D0 zurück. Es wird also nach dem Ablauf eines Spiels auf das Betätigen einer Taste gewartet. Mit dem Unterprogrammaufruf „CLR" wird anschließend der Bildschirm gelöscht, und dann beginnt das Spiel von vorne.

Aufgaben:

1. Ergänzen Sie das Struktogramm, so daß es nun dem Programm exakt entspricht. Es soll dann auch die Wiederholung der Tasteneingabe am Schluß enthalten.
2. Wie kann man die Geschwindigkeit des Spielablaufs beeinflussen?
3. Wie kann man die Größe der Landefähre einfach bestimmen?
4. Was passiert, wenn man den Warte-Block wegläßt? (Ausprobieren, indem man die Sprünge „BEQ WARTE20MS" und „DBRA D1,WARTE20MS" herauslöscht).

Anschluß eines Joysticks

Joysticks sind bei Homecomputern für Spielprogramme gebräuchlich. Ein solcher Joystick besitzt im Inneren vier kleine Schalter. Je nach Bewegungsrichtung wird einer oder werden zwei dieser Schalter betätigt. Die *Abb. 3.4.9* zeigt ein Schema. Wenn wir die Schalter unseres Tastenfeldes durch einen solchen Joystick ersetzen, so können wir damit unsere Mondlandefähre steuern. Die Joysticks besitzten normalerweise noch eine Feuertaste, mit der sich zusätzliche Effekte erzielen lassen.

Die meisten dieser Joysticks sind für einen Anschluß über eine 9-polige Stiftleiste vorgesehen, die glücklicherweise die gleiche Belegung aufweist. Man kann den Joystick somit einfach an die IOE-Baugruppe anschließen. *Abb. 3.4.10* zeigt die nötigen Verbindungsleitungen. Links sind die Nummern der Buchse und des Stiftverbinders eingetragen. Rechts die Nummern der einzelnen Bits. *Abb. 3.4.11* zeigt ein mögliches Verdrahtungschema. Dabei wird eine doppelreihige Buchsenleiste oder ein Flachbandkabel mit einer Anpreßbuchse auf der IOE-Seite verwendet.

Die Bitanordnung am Port entspricht den bei Home-Computern gebräuchlichen Zuordnungen und lautet:
Bit 0 Joystick nach oben
Bit 1 Joystick nach unten
Bit 2 Joystick nach links
Bit 3 Joystick nach rechts
Bit 4 Feuertaste am Joystick.

Unser Mondlandeprogramm läßt sich leicht auf die Verwendung eines Joysticks umschreiben. *Abb. 3.4.12* zeigt das Listing. Man könnte zunächst einmal einfach die Bitzuordnungen bei den „EQU"-Anweisungen verändern, da die einzelnen Bits nun eine vertauschte Rolle besitzen. Hier wurde aber noch mehr geändert. Diesmal sind nicht die Bitmasken zugeordnet, sondern die Bitnummern. Im Hauptprogramm „START" wird dann nicht durch die Und-Verknüpfung der einzelne Wert eines Bits abgefragt, sondern mit dem Befehl „BTST".

Der Assemblerbefehl BTST

Der Befehl „BTST #2,D0" prüft z. B. Bit 2 des Registers D0. Ist das Bit gleich 0, so wird das Z-Flag gesetzt. Ein nachfolgender Sprung „BNE" springt somit genau dann, wenn das entsprechende Bit gesetzt war. Der Vorteil des „BTST"-Befehls liegt darin, daß er nur die Flags setzt, nicht aber den Registerinhalt verändert. Man kann dadurch die Bits nacheinander abfragen, ohne den Wert jedesmal neu in das Register laden zu müssen, wie das beim „AND" nötig war.

Eine weitere Ergänzung ist der Einbau des Unterprogramms LASER und LASER-EXEC. Damit tritt nun die Feuertaste am Joystick in Aktion. Wenn man sie drückt, so wird gemäß *Abb. 3.4.13* ein Laser über den Bildschirm wandern.

Zunächst fällt uns im Listing die neu eingebaute Abfrage an der Marke TST4 für die Feuertaste auf. Wurde diese betätigt, so wird zunächst festgestellt, ob sich noch ein früher abgefeuerter Laserstrahl auf dem Bildschirm befindet. Ist dies nicht der Fall, so

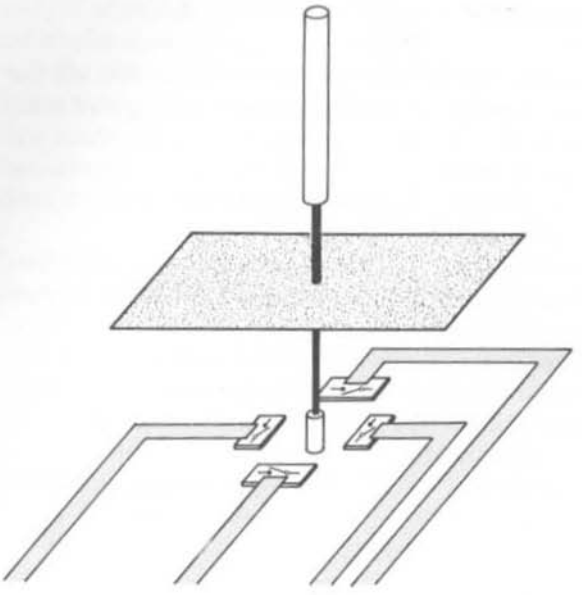


Abb. 3.4.9
Schematischer Aufbau
eines Joysticks

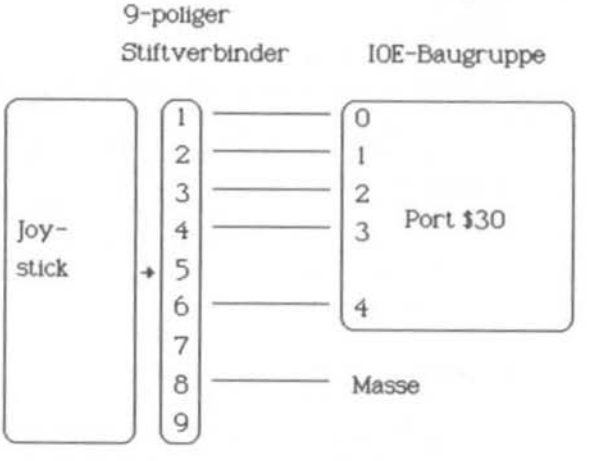


Abb. 3.4.10
Schematischer Anschluß
an die IOE-Baugruppe

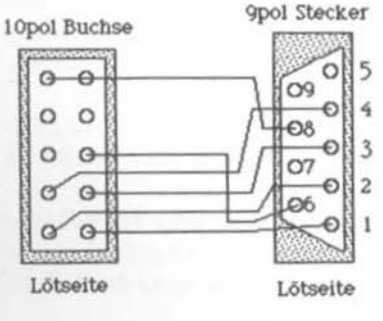


Abb. 3.4.11 Verdrahtungsschema

```
Rolf-D.Klein 68000/08 Assembler 4.3 (C) 1984, Seite 1

009C00
009C00                    ;* MONDLANDEPROGRAMM MIT JOYSTICK
009C00                    ;
009C00                    ORG $9C00
009C00
009C00                    * VEREINBARUNGSBAUSTEIN
009C00
= FFFFFF30                TASTEN EQU $FFFFFF30
= 00000003                RECHTS EQU 3
= 00000002                LINKS EQU 2
= 00000001                RAUF EQU  1
= 00000000                RUNTER EQU 0
= 00000004                FEUER EQU 4
009C00
009C00                    ERFOLGTXT:
009C00  4775742067656C     DC.B 'Gut gelandet',0
009C07  616E64657400
009C0D                    BRUCHTXT:
009C0D  42727563686C61     DC.B 'Bruchlandung',0
009C14  6E64756E6700
009C1A                    FAEHRE:
009C1A  00 01 01 00 00     DC.B 0,1,1,0,0,0,0,0,7,7,0,4,3,3
009C1F  00 00 00 07 07
009C24  00 04 03 03
009C28  02 04 04 04 04     DC.B 2,4,4,4,4,4,6,2,0
009C2D  04 06 02 00
009C31  03 02 02 02 01     DC.B 3,2,2,2,1,0,0,0,7,6,6,6,5,10
009C36  00 00 00 07 06
009C3B  06 06 05 0A
009C3F
009C3F  00                  DS 0
009C40
009C40  0000               X: DC.W 0
009C42  0000               Y: DC.W 0
009C44  0000               DX: DC.W 0
009C46  0000               DY: DC.W 0
009C48  0000               TREIBST: DC.W 0
009C4A
009C4A  0000               HX: DC.W 0
009C4C  0000               HY: DC.W 0
009C4E  0000               DHX: DC.W 0
009C50  0000               DHY: DC.W 0
009C52
009C52
009C52                    * UNTERPROGRAMMBAUSTEIN
009C52
009C52                    ;
009C52                    ;
009C52                    MONDFLAECHE:
009C52  4241               CLR D1
009C54  4242               CLR D2
009C56  363C 005A          MOVE #90,D3
```

zu Abb. 3.4.12

```
Rolf-D.Klein 68000/08 Assembler 4.3 (C) 1984, Seite 2

009C5A  4EB9 0000131A    JSR @SET
009C60  4EB9 00001112    JSR @HIDE
009C66  383C 0031        MOVE #50-1,D4
009C6A                  MLP1:
009C6A  4EB9 00009C76    JSR KRATER
009C70  51CC FFF8        DBRA D4,MLP1
009C74  4E75             RTS
009C76
009C76                  KRATER:
009C76  303C FFD3        MOVE #-45,D0
009C7A  4EB9 000012F6    JSR @DREHE
009C80  303C 000A        MOVE #10,D0
009C84  4EB9 00001288    JSR @SCHREITE
009C8A  303C FFD3        MOVE #-45,D0
009C8E  4EB9 000012F6    JSR @DREHE
009C94  303C 0006        MOVE #6,D0
009C98  4EB9 00001288    JSR @SCHREITE
009C9E  303C FFD3        MOVE #-45,D0
009CA2  4EB9 000012F6    JSR @DREHE
009CA8  303C 000A        MOVE #10,D0
009CAC  4EB9 00001288    JSR @SCHREITE
009CB2  303C FF1F        MOVE #-45-180,D0
009CB6  4EB9 000012F6    JSR @DREHE
009CBC  4E75             RTS
009CBE                  ;
009CBE                  LASER:   * LASERSTRAHL-FORM
009CBE  3239 00009C4A    MOVE HX,D1
009CC4  3439 00009C4C    MOVE HY,D2
009CCA  4EB9 00000EE2    JSR @MOVETO
009CD0  0641 0019        ADD #25,D1
009CD4  4EB9 00000F4C    JSR @DRAWTO
009CDA  4E75             RTS
009CDC
009CDC                  LASEREXEC: * LASER NEU AUSGEBEN
009CDC                  * LASER
009CDC  4A79 00009C4C    TST HY
009CE2  6700 0036        BEQ WDH12
009CE6                  * LOESCHEN ALTE POS
009CE6  4EB9 00000D6E    JSR @ERAPEN
009CEC  6100 FFD0        BSR LASER
009CF0  0679 000A        ADD #10,HY
009CF4  00009C4A
009CF8  4EB9 00000D7C    JSR @SETPEN
009CFE  6100 FFBE        BSR LASER
009D02  0C79 0200        CMP #512,HX
009D06  00009C4A
009D0A  6500 000E        BCS WDH12                    zu Abb. 3.4.12
009D0E  4279 00009C4C    CLR HY
009D14  4279 00009C4A    CLR HX * STOP
009D1A                  WDH12:
009D1A                  *
009D1A  4E75             RTS
009D1C
```

```
Rolf-D.Klein 68000/08 Assembler 4.3 (C) 1984, Seite 3

009D1C                  * PROGRAMMBAUSTEIN
009D1C
009D1C                  START:
009D1C  6100 FF34        BSR MONDFLAECHE
009D20  33FC 0064        MOVE #100,X
009D24  00009C40
009D28  33FC 00C8        MOVE #200,Y
009D2C  00009C42
009D30  33FC 0003        MOVE #3,DX
009D34  00009C44
009D38  33FC 0000        MOVE #0,DY
009D3C  00009C46
009D40  33FC 012C        MOVE #300,TREIBST
009D44  00009C48
009D48                  WDH1:
009D48  41F9 00009C1A    LEA FAEHRE,A0
009D4E  303C 0003        MOVE #3,D00
009D52  3239 00009C40    MOVE X,D1
009D58  3439 00009C42    MOVE Y,D2
009D5E  4EB9 0000328C    JSR @FIGUR
009D64  4A79 00009C48    TST TREIBST
009D6A  6F00 006C        BLE NOTEST
009D6E  1039 FFFFFF30    MOVE.B TASTEN,D0
009D74  0800 0003        BTST #RECHTS,D0
009D78  6606             BNE.S TST1
009D7A  5679 00009C44    ADDQ #3,DX
009D80                  TST1:
009D80  0800 0002        BTST #LINKS,D0
009D84  6606             BNE.S TST2
009D86  5779 00009C44    SUBQ #3,DX
009D8C                  TST2:
009D8C  0800 0001        BTST #RAUF,D0
009D90  6606             BNE.S TST3
009D92  5679 00009C46    ADDQ #3,DY
009D98                  TST3:
009D98  0800 0000        BTST #RUNTER,D0
009D9C  6606             BNE.S TST4
009D9E  5779 00009C46    SUBQ #3,DY
009DA4                  TST4:
009DA4  0800 0004        BTST #FEUER,D0
009DA8  662E             BNE.S TST5
009DAA  4A79 00009C4C    TST HY
009DB0  6600 0026        BNE TST5
009DB4  33F9 00009C40    MOVE X,HX
009DBA  00009C4A
009DBE  33F9 00009C42    MOVE Y,HY
009DC4  00009C4C
009DC8  0679 0014        ADD #20,HX
009DCC  00009C4A
009DD0  0679 0014        ADD #20,HY
009DD4  00009C4C
009DD8                  TST5:
009DD8                  NOTEST:
```

zu Abb. 3.4.12

```
Rolf-D.Klein 68000/08 Assembler 4.3 (C) 1984, Seite 4

009DD8  5379 00009C46     SUBQ #1,DY
009DDE  3639 00009C44     MOVE DX,D3
009DE4  D779 00009C40     ADD D3,X
009DEA  3639 00009C46     MOVE DY,D3
009DF0  D779 00009C42     ADD D3,Y
009DF6  5379 00009C48     SUBQ #1,TREIBST
009DFC  323C 0004         MOVE #5-1,D1
009E00                   WARTE20MS:
009E00  4EB9 00000E38     JSR @SYNC
009E06  6700 FFF8         BEQ WARTE20MS
009E0A  3F01              MOVE D1,-(A7)
009E0C  6100 FECE         BSR LASEREXEC
009E10  321F              MOVE (A7)+,D1
009E12  51C9 FFEC         DBRA D1,WARTE20MS
009E16  0C79 0005         CMP #5,Y
009E1A  00009C42
009E1E  6E00 FF28         BGT WDH1
009E22  303C 0003         MOVE #3,D0
009E26  3239 00009C40     MOVE X,D1
009E2C  3439 00009C42     MOVE Y,D2
009E32  4EB9 0000328C     JSR @FIGUR
009E38  3039 00009C44     MOVE DX,D0
009E3E  6A02              BPL.S FF1
009E40  4440              NEG D0
009E42                   FF1:
009E42  0C40 0002         CMP #2,D0
009E46  6C00 0020         BGE BRUCH
009E4A  3039 00009C46     MOVE DY,D0
009E50  6A00 0004         BPL FF2
009E54  4440              NEG D0
009E56                   FF2:
009E56  0C40 0004         CMP #4,D0
009E5A  6C00 000C         BGE BRUCH
009E5E  41F9 00009C00     LEA ERFOLGTXT,A0
009E64  6000 0008         BRA ENDE
009E68                   BRUCH:
009E68  41F9 00009C0D     LEA BRUCHTXT,A0
009E6E                   ENDE:
009E6E  303C 0033         MOVE #$33,D0
009E72  4241              CLR D1
009E74  343C 0082         MOVE #130,D2
009E78  4EB9 00001386     JSR @WRITE
009E7E  4EB9 00000A00     JSR @CI
009E84  4EB9 00000DA0     JSR @CLR
009E8A  6000 FE90         BRA START
009E8E
009E8E                    END

008CE8  Ende-Symboltabelle
```

Abb. 3.4.12 Die Mondlandung mit Joystick-Steuerung

Abb. 3.4.13
Die Landefähre mit abgefeuertem Schuß

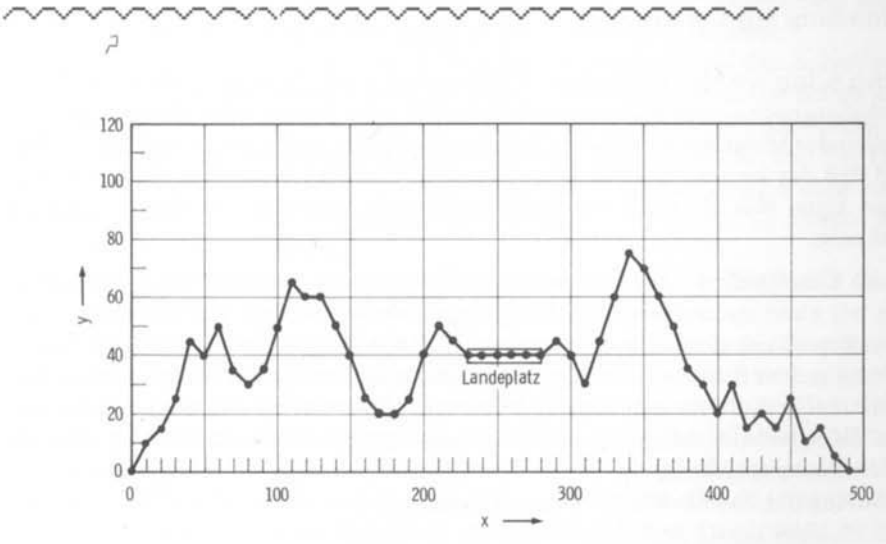


Abb. 3.4.14 Eine Gebirgslandschaft für die Mondlandung

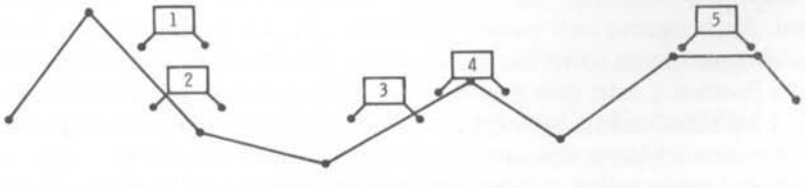


Abb. 3.4.15 Verschiedene Positionen der Landefähre bezüglich der Oberfläche

wird die Variable HY mit einem Wert ungleich null belegt. Das neu eingebaute Unterprogramm LASEREXEC gibt dann mit Hilfe des Unterprogrammes LASER ein Stück Laserstrahl auf den Bildschirm aus, wenn die Variable HY mit einem Wert ungleich null belegt wurde.

Im Unterprogramm „LASEREXEC" wird zunächst der Laser bei der alten Position gelöscht; dies geschieht durch „ERAPEN" und „BSR LASER", und dann wird der Laser an der neuen Position ausgegeben. Mit „SETPEN" wird der Stift wieder auf Schreiben geschaltet. Verläßt der Laserstrahl den Bildschirm, so wird HY wieder auf null gesetzt.

Die Ausgabe des Lasers erfolgt alle 20 Millisekunden, der Aufruf steht in der Warteschleife. Die Landefähre wird ja nur alle 100 Millisekunden neu ausgegeben, es gibt somit keine andere Stelle im Programm, die einen Aufruf alle 20 Millisekunden gestattet. Der restliche Ablauf entspricht dem Mondlandeprogramm wie gehabt.

Mondlandung mit Gebirge

Eigentlich haben wir bis hierher unser Pflichtenheft noch nicht ganz erfüllt. Dort steht die Forderung nach einem ebenen Landeplatz. Wie wir wissen, ist der Mond nicht eben. Wir wollen das Mondlandeprogramm darum so gestalten, daß auch ein Gebirge sichtbar ist, und daß das Programm Kollisionen der Mondlandefähre mit dem Gebirge richtig erkennen kann. Nur ein dafür vorgesehener ebener Landeplatz soll für die Landung geeignet sein.

Für die Konstruktion der Gebirgslandschaft, wie sie in der *Abb. 3.4.14* gezeigt ist, werden wir nicht mehr unsere Schildkrötengraphik verwenden. Wir geben die Landschaft in Koordinatenform ein. Unsere neue Methode geht von einem Ansteigen der x-Koordinate in 10er Schritten aus. Diese Schrittweite ist für unser Vorhaben klein genug, außerdem ist es dadurch möglich, nicht zu viele Werte der y-Achse abspeichern zu müssen. Nachdem die x-Koordinate fest vorgegeben ist, sind nur noch die y-Werte in einer Tabelle zu speichern.

Der Anfang der Tabelle könnte dann so aussehen:

```
dc.w 10,15,25,45 ...
```

Der Landeplatz befindet sich etwa in der Mitte des Gebirges. Er muß später im Programm gesondert behandelt werden. Denn nur dort kann die Landefähre sicher landen.

Bisher konnte man im Programm die Abfrage der Höhe über Grund recht einfach gestalten. Dort stand der Befehl „CMP #5,Y" im Programm. Nun ist aber die Höhe von der x-Koordinate abhängig. Der Vergleich muß also von der x-Position der Fähre abhängen. Auch ergeben sich weitere Probleme. *Abb. 3.4.15* zeigt ein paar Positionen, die die Fähre annehmen kann. Die Position 1 ist erlaubt, die Fähre berührt das Gebirge nicht. Die Position 2 zeigt eine Kollision des linken Landefußes mit dem Gebirge. Die Position 3 kollidiert rechts, bei 4 berührt die Fähre in der Mitte die Spitze des Berges, und die Position 5 könnte eine Landung auf der ebenen Fläche sein.

Die y-Koordinaten sollen in 10er Schritten im Speicher vorhanden sein. Für Zwischenwerte muß aber auch eine jeweilige Höhe berechnet werden. Dazu zeigt *Abb.*

3.4.16 ein Schema. Das Gebirge setzt sich aus lauter solchen kleinen Abschnitten zusammen. Die Strecke zwischen X1;Y1 und X2;Y2 bildet die Schräge des Gebirges. Die Fähre besitzt die Koordinaten x;y. Um festzustellen, ob die Fähre mit dem Gebirge kollidiert, muß man die y-Koordinate der Fähre mit der y-Koordinate des Schnittpunktes S auf der Strecke vergleichen. Ist Y-Faehre größer als Y-Schnittpunkt, findet keine Kollision statt.

Nun genügt die x;y-Position der Fähre des linken Landefußes nicht allein, man müßte eigentlich alle Punkte der Fähre zum Vergleich heranziehen. Wir wollen in unserem Programm aber nur noch die Position des rechten Beines hinzuziehen und damit die beiden tiefsten Punkte der Fähre zum Vergleich verwenden. Damit kann man nicht alle Fälle exakt erfassen; die Position 4 der Landefähre aus Abb. 3.4.15 führt erst dann zur Kollision, wenn eines der beiden Beine das Gebirge berührt. Die Spitze des Gebirges kollidiert hier aber nicht mit der Unterseite der Landefähre. Bei dem einfachen Vergleich mit dem Schnittpunkt kann man auch keine Überhänge verwenden, dß die Fähre immer oberhalb von Gebirgslinien liegen muß. Durch unsere Graphikwahl mit festen Werten für die x-Richtung sind ohnehin keine Überhänge darstellbar.

Abb. 3.4.17 zeigt ein Schema zur Ableitung der Gleichung für den Schnittpunkt S. Dazu wird der mathematische Satz aus *Abb. 3.4.18* verwendet. Die Seiten a und b verhalten sich zueinander wie die Seiten c und d. Daraus ergibt sich die Formel in *Abb.* 3.4.19.Der y-Abschnitt des Punktes S ergibt sich aus der Summe YSA und Y2. Y2 ist die eine Koordinate des rechten Punktes der Strecke (X1;Y1)-(X2;Y2). YSA kann man dann berechnen. Die Differenz Δx beträgt bei uns genau 10, da wir 10 Punkte Abstand zwischen den Punkten haben.

Die *Abb. 3.4.20* zeigt das Programmlisting. Bei der Marke LANDSCHAFT sind die y-Koordinaten für die Mondoberfläche in einer Tabelle aufgelistet.

Das Unterprogramm CHECKHOEHE liefert den Vergleich in Abhängigkeit der x-Koordinate. Das Ergebnis ist 0, wenn die Landefähre an einem der beiden Randpunkte der Landebeine unterhalb der Gebirgsoberfläche liegt. Sonst wird der Wert $FFFF im Register D0 geliefert. Dieses Unterprogramm wird nun anstelle des alten „CMP"-Befehls im Hauptprogramm eingebaut. Zusätzlich enthält das Hauptprogramm zwei Abfragen, ob sich die Fähre im Landegebiet befindet. Dabei ist XMIN=230 und XMAX=280. Die Breite der Landefähre ist als Konstante „FBREITE" definiert. Damit wird die Position des rechten Landebeins berechnet.

Die *Abb. 3.4.21* zeigt den Bildschirm nach einer geglückten Landung. Die Landung ist jetzt nicht mehr so einfach, es gehört schon viel Geschick dazu. Viel Spaß.

Aufgaben:

1. Was bewirken die Befehle „MULU #10,D0" und „DIVU #10,D0", wenn sie nacheinander ausgeführt werden?
2. Zeichnen Sie ein Struktogramm zum Unterprogramm „CHECKHOEHE".
3. In der Abb. 3.4.21 sieht man, daß die erfolgreiche Landung ein klein wenig unter der Oberfläche stattfindet. Woran liegt das? Versuchen Sie, das Programm zu ändern, so daß eine Landung exakt auf der Oberfläche möglich ist.

Abb. 3.4.16
Schematische Lage
der Fähre

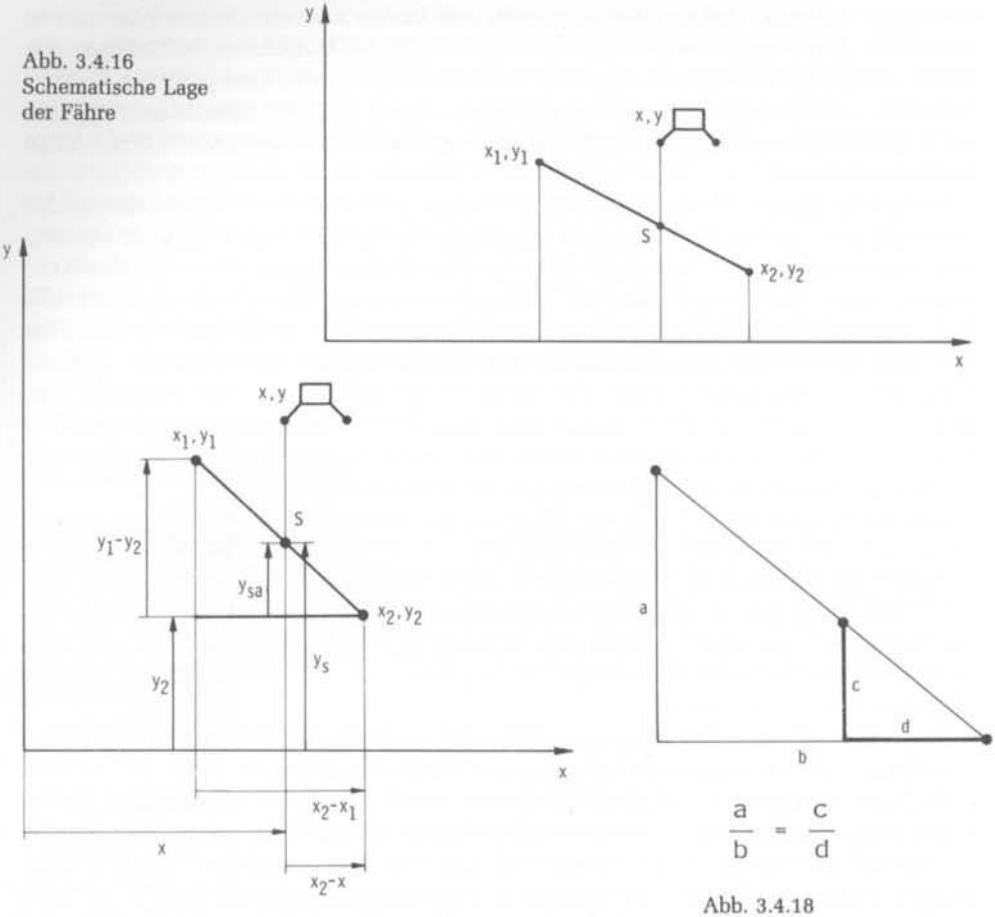


Abb. 3.4.17 Ableitung der Gleichung

$$\frac{a}{b} = \frac{c}{d}$$

Abb. 3.4.18
Die Seiten im Dreieck

$$Y_S = Y_{Sa} + Y_2$$

$$\frac{Y_1 - Y_2}{\Delta X} = \frac{Y_{Sa}}{X_2 - X}$$

$$\frac{Y_1 - Y_2}{\Delta X} * (X_2 - X) = Y_{Sa}$$

Abb. 3.4.19
Die Formeln zur Mondlandung

```
Rolf-D.Klein 68000/08 Assembler 4.3 (C) 1984, Seite 1


009C00
009C00                    ;* MONDLANDEPROGRAMM MIT JOYSTICK
009C00                    ;* UND MONDOBERFLAECHE MIT EBENEM LANDEPLATZ
009C00
009C00                    ORG $9C00
009C00
009C00                    * VEREINBARUNGSBAUSTEIN
009C00
= FFFFFF30                TASTEN EQU $FFFFFF30
= 00000003                RECHTS EQU 3
= 00000002                LINKS EQU 2
= 00000001                RAUF EQU  1
= 00000000                RUNTER EQU 0
= 00000004                FEUER EQU 4
009C00
009C00                    ERFOLGTXT:
009C00  4775742067656C     DC.B 'Gut gelandet',0
009C07  616E64657400
009C0D                    BRUCHTXT:
009C0D  42727563686C6.     DC.B 'Bruchlandung',0
009C14  6E64756E6700
009C1A                    FAEHRE:
009C1A  00 01 01 00 00     DC.B 0,1,1,0,0,0,0,0,7,7,0,4,3,3
009C1F  00 00 00 07 07
009C24  00 04 03 03
009C28  02 04 04 04 04     DC.B 2,4,4,4,4,4,4,6,2,0
009C2D  04 06 02 00
009C31  03 02 02 02 01     DC.B 3,2,2,2,1,0,0,0,7,6,6,6,5,10
009C36  00 00 00 07 06
009C3B  06 06 05 0A
009C3F  00                 DS 0
009C40
= 00000021                FBREITE EQU 11*3   * BREITE DER FAEHRE
009C40
009C40                    LANDSCHAFT:    * Y-KOORDINATEN, DER FLAECHE
009C40                                   * DIE X-KOORDINATEN LAUFEN IN 10ER SCHRITTEN
009C40  0000 000A 000F     DC.W 0,10,15,25,45,40,50,35,30,35,50,65,60,60,50,40,25,20,20
009C46  0019 002D 0028
009C4C  0032 0023 001E
009C52  0023 0032 0041
009C58  003C 003C 0032
009C5E  0028 0019 0014
009C64  0014
009C66  0019 0028 0032     DC.W 25,40,50,45,40,40,40,40,40,40,45,40,30,45,60,75
009C6C  002D 0028 0028
009C72  0028 0028 0028
009C78  0028 002D 0028
009C7E  001E 002D 003C
009C84  004B
009C86  0046 003C 0032     DC.W 70,60,50,35,30,20,30,15,20,15,25,10,15,5,0,0
009C8C  0023 001E 0014
009C92  001E 000F 0014
```

zu Abb. 3.4.20

```
Rolf-D.Klein 68000/08 Assembler 4.3 (C) 1984, Seite 2

009C98  000F 0019 000A
009C9E  000F 0005 0000
009CA4  0000
= 00000033              ANZLAND EQU (*-LANDSCHAFT)/2  * HALBE WEGEN WORT
009CA6
009CA6  0000            X: DC.W 0
009CA8  0000            Y: DC.W 0
009CAA  0000            DX: DC.W 0
009CAC  0000            DY: DC.W 0
009CAE  0000            TREIBST: DC.W 0
009CB0
009CB0  0000            HX: DC.W 0
009CB2  0000            HY: DC.W 0
009CB4  0000            DHX: DC.W 0
009CB6  0000            DHY: DC.W 0
009CB8
009CB8
009CB8                  * UNTERPROGRAMMBAUSTEIN
009CB8
009CB8                  ;
009CB8                  ;
009CB8                  MONDFLAECHE:
009CB8  4241             CLR D1
009CBA  4242             CLR D2
009CBC  4EB9 00000EE2    JSR @MOVETO   * DORT ANFANGEN
009CC2  41F9 00009C40    LEA LANDSCHAFT,A0
009CC8  363C 0032        MOVE #ANZLAND-1,D3
009CCC                  MONDLP:
009CCC  3418             MOVE (A0)+,D2
009CCE  4EB9 00000F4C    JSR @DRAWTO
009CD4  0641 000A        ADD #10,D1
009CD8  51CB FFF2        DBRA D3,MONDLP
009CDC  4E75             RTS
009CDE
009CDE                  ;
009CDE                  LASER:   * LASERSTRAHL-FORM
009CDE  3239 00009CB0    MOVE HX,D1
009CE4  3439 00009CB2    MOVE HY,D2
009CEA  4EB9 00000EE2    JSR @MOVETO
009CF0  0641 0019        ADD #25,D1
009CF4  4EB9 00000F4C    JSR @DRAWTO
009CFA  4E75             RTS
009CFC
009CFC                  LASEREXEC: * LASER NEU AUSGEBEN
009CFC                  * LASER
009CFC  4A79 00009CB2    TST HY
009D02  6700 0036        BEQ WDH12
009D06                  * LOESCHEN ALTE POS
009D06  4EB9 00000D6E    JSR @ERAPEN
009D0C  6100 FFD0        BSR LASER                    zu Abb. 3.4.20
009D10  0679 000A        ADD #10,HX
009D14  00009CB0
009D18  4EB9 00000D7C    JSR @SETPEN
```

```
Rolf-D.Klein 68000/08 Assembler 4.3 (C) 1984, Seite 3

009D1E  6100 FFBE        BSR LASER
009D22  0C79 0200        CMP #512,HX
009D26  00009CB0
009D2A  6500 000E        BCS WDH12
009D2E  4279 00009CB2    CLR HY
009D34  4279 00009CB0    CLR HX * STOP
009D3A                   WDH12:
009D3A                   *
009D3A  4E75              RTS
009D3C
009D3C                   CHECKHOEHE:    * Y IN ABHAENIGKEIT VON X
009D3C                                  * D0=FFFF, WENN OK, SONST 0
009D3C  3239 00009CA6     MOVE X,D1
009D42  6B00 0072         BMI NOTOK     * AUSSERHALB DES BILDSCHIRMS
009D46  0C41 0200         CMP #512,D1
009D4A  6E00 006A         BGT NOTOK     * IMMER BRUCH
009D4E  3439 00009CA8     MOVE Y,D2
009D54  6100 001C         BSR CHECK1    * ZUERST LINKER AUFSETZPUNKT
009D58  6700 005C         BEQ NOTOK     * NEIN, DANN UNTER OBERFLAECHE
009D5C  3239 00009CA6     MOVE X,D1
009D62  3439 00009CA8     MOVE Y,D2
009D68  0641 0021         ADD #FBREITE,D1  * NUN FUER RECHTE SEITE PRUEFEN
009D6C  6100 0004         BSR CHECK1
009D70  4E75              RTS
009D72
009D72                   CHECK1:       * X=D1, Y=D2 , PRUEFT HOEHE
009D72  3601              MOVE D1,D3   * TABELLENEINTRAG FINDEN
009D74  48C3              EXT.L D3     * DAZU DURCH 10 DIVIDIEREN
009D76  86FC 000A         DIVU #10,D3 * UND ALS WORT HOLEN
009D7A  43F9 00009C40     LEA LANDSCHAFT,A1
009D80  D2C3              ADDA.W D3,A1
009D82  D2C3              ADDA.W D3,A1 * 2* ADDIEREN, DA WORT
009D84  3611              MOVE (A1),D3 * Y1
009D86  9669 0002         SUB 2(A1),D3 * Y1-Y2
009D8A  3801              MOVE D1,D4
009D8C  48C4              EXT.L D4
009D8E  88FC 000A         DIVU #10,D4   * ERST MAL X1 BERECHNEN
009D92  C8FC 000A         MULU #10,D4
009D96  0644 000A         ADD #10,D4    * DA 10ER ABSTAND  =X2
009D9A  9841              SUB D1,D4     * IST X2-X
009D9C  C7C4              MULS D4,D3    * (Y2-Y1)*(X2-X)
009D9E  87FC 000A         DIVS #10,D3   * = YSA
009DA2  D669 0002         ADD 2(A1),D3 * IST YS  HOEHE DES SCHNITTPUNKTES
009DA6  B642              CMP D2,D3
009DA8  6F00 0006         BLE OKCHECK
009DAC  6000 0008         BRA NOTOK
009DB0
009DB0                   OKCHECK:
009DB0  303C FFFF         MOVE #$FFFF,D0
009DB4  4E75              RTS                          zu Abb. 3.4.20
009DB6                   NOTOK:
009DB6  4240              CLR D0
009DB8  4E75              RTS
```

```
Rolf-D.Klein 68000/08 Assembler 4.3 (C) 1984, Seite 4

009DBA
009DBA
009DBA
009DBA                    * PROGRAMMBAUSTEIN
009DBA
009DBA                    START:
009DBA  6100 FEFC          BSR MONDFLAECHE
009DBE  33FC 0064          MOVE #100,X
009DC2  00009CA6
009DC6  33FC 00C8          MOVE #200,Y
009DCA  00009CA8
009DCE  33FC 0003          MOVE #3,DX
009DD2  00009CAA
009DD6  33FC 0000          MOVE #0,DY
009DDA  00009CAC
009DDE  33FC 012C          MOVE #300,TREIBST
009DE2  00009CAE
009DE6                    WDH1:
009DE6  41F9 00009C1A      LEA FAEHRE,A0
009DEC  303C 0003          MOVE #3,D0
009DF0  3239 00009CA6      MOVE X,D1
009DF6  3439 00009CA8      MOVE Y,D2
009DFC  4EB9 0000328C      JSR @FIGUR
009E02  4A79 00009CAE      TST TREIBST
009E08  6F00 006C          BLE NOTEST
009E0C  1039 FFFFFF30      MOVE.B TASTEN,D0
009E12  0800 0003          BTST #RECHTS,D0
009E16  6606               BNE.S TST1
009E18  5679 00009CAA      ADDQ #3,DX
009E1E                    TST1:
009E1E  0800 0002          BTST #LINKS,D0
009E22  6606               BNE.S TST2
009E24  5779 00009CAA      SUBQ #3,DX
009E2A                    TST2:
009E2A  0800 0001          BTST #RAUF,D0
009E2E  6606               BNE.S TST3
009E30  5679 00009CAC      ADDQ #3,DY
009E36                    TST3:
009E36  0800 0000          BTST #RUNTER,D0
009E3A  6606               BNE.S TST4
009E3C  5779 00009CAC      SUBQ #3,DY
009E42                    TST4:
009E42  0800 0004          BTST #FEUER,D0
009E46  662E               BNE.S TST5
009E48  4A79 00009CB2      TST HY
009E4E  6600 0026          BNE TST5
009E52  33F9 00009CA6      MOVE X,HX
009E58  00009CB0
009E5C  33F9 00009CA8      MOVE Y,HY
009E62  00009CB2
009E66  0679 0014          ADD #20,HX
009E6A  00009CB0
009E6E  0679 0014          ADD #20,HY
```

zu Abb. 3.4.20

```
Rolf-D.Klein 68000/08 Assembler 4.3 (C) 1984, Seite 5

009E72  00009CB2
009E76                  TST5:
009E76                  NOTEST:
009E76  5379 00009CAC    SUBQ #1,DY
009E7C  3639 00009CAA    MOVE DX,D3
009E82  D779 00009CA6    ADD D3,X
009E88  3639 00009CAC    MOVE DY,D3
009E8E  D779 00009CA8    ADD D3,Y
009E94  5379 00009CAE    SUBQ #1,TREIBST
009E9A  323C 0004        MOVE #5-1,D1
009E9E                  WARTE20MS:
009E9E  4EB9 00000E38    JSR @SYNC
009EA4  6700 FFF8        BEQ WARTE20MS
009EA8  3F01             MOVE D1,-(A7)
009EAA  6100 FE50        BSR LASEREXEC
009EAE  321F             MOVE (A7)+,D1
009EB0  51C9 FFEC        DBRA D1,WARTE20MS
009EB4  6100 FE86        BSR CHECKHOEHE   * HIER IN ABH. VON X PRUEFEN
009EB8  6600 FF2C        BNE WDH1         * WENN <> 0 DANN OK
009EBC  303C 0003        MOVE #3,D0       * SONST AM BODEN,
009EC0  3239 00009CA6    MOVE X,D1        * FIGUR ABER ERST MAL DARSTELLEN
009EC6  3439 00009CA8    MOVE Y,D2
009ECC  4EB9 000032BC    JSR @FIGUR
009ED2  4EB9 00003286    JSR @SETFIG
009ED8  3039 00009CA6    MOVE X,D0        * PRUEFEN OB IM LANDEGEBIET
009EDE  0C40 00E6        CMP #230,D0      * BEI UNS XMIN=230
009EE2  6D00 003E        BLT BRUCH        * NEIN, DANN BRUCH
009EE6  0640 0021        ADD #FBREITE,D0  * ODER RECHTS ZU WEIT ?
009EEA  0C40 0118        CMP #280,D0
009EEE  6E00 0032        BGT BRUCH        * DANN AUCH BRUCH
009EF2  3039 00009CAA    MOVE DX,D0       * BESCHLEUNIGUNG TESTEN
009EF8  6A02             BPL.S FF1
009EFA  4440             NEG D0
009EFC                  FF1:
009EFC  0C40 0002        CMP #2,D0
009F00  6C00 0020        BGE BRUCH
009F04  3039 00009CAC    MOVE DY,D0
009F0A  6A00 0004        BPL FF2
009F0E  4440             NEG D0
009F10                  FF2:
009F10  0C40 0004        CMP #4,D0
009F14  6C00 000C        BGE BRUCH
009F18  41F9 00009C00    LEA ERFOLGTXT,A0
009F1E  6000 0008        BRA ENDE
009F22                  BRUCH:
009F22  41F9 00009C0D    LEA BRUCHTXT,A0
009F28                  ENDE:
009F28  303C 0033        MOVE #$33,D0
009F2C  4241             CLR D1
009F2E  343C 0082        MOVE #130,D2
009F32  4EB9 00001386    JSR @WRITE
009F38  4EB9 00000A00    JSR @CI
009F3E  4EB9 00000DA0    JSR @CLR
009F44  6000 FE74        BRA START
009F48
009F48                    END

008D8A  Ende-Symboltabelle
```

Abb. 3.4.20
Mondlandeprogramm mit Gebirge

4. *Abb. 3.4.22* zeigt ein Programm zur Simulation des Fluges eines Balles. Der Ball fällt von links oben ins Bildfeld und wird dann am Boden reflektiert. Dabei wird auch eine Dämpfung des Balles durch einen nicht elastischen Stoß simuliert. Analysieren Sie das Programm, und verbessern Sie ggf. den Algorithmus. Die Formeln sind dabei ähnlich wie bei der Mondlandung. Das Programm kann man auch noch weiter ausbauen, wenn man Hindernisse im Programm angibt oder eine schräge Ebene einführt.

3.5 Zeitmessung

Die genaue Zeitnahme, etwa bei einem Mondlandewettbewerb, war bisher noch nicht möglich. Die Zeitnahme mit einer Stoppuhr ist nicht sehr genau, außerdem, wozu haben wir einen Computer? Aufgabe dieser Folge ist es also, eine Zeitausgabe im Bildschirm einzublenden und automatisch die Zeit zwischen Start und Stop des Mondlande-Spiels anzuzeigen.

Dazu lösen wir zunächst eine Teilaufgabe. Wie konstruiert man eine Uhr per Software? Sehen wir uns zum Vergleich die Funktion einer gewöhnlichen Uhr an. Da gibt es zunächst einen Taktgeber. Das kann ein Pendel oder eine Unruhe sein, bei modernen Uhren findet ein Quarzgeber Anwendung. In unserem Computer befinden sich mehrere Quarz-Taktgeber. Einmal auf der CPU-Baugruppe und dann auf der GDP-Karte. Man kann eine Zeitmessung dadurch erreichen, daß man die Takte abzählt. Am einfachsten läßt sich der Taktgeber auf der GDP-Baugruppe dazu verwenden. Zunächst einmal schwingt der Quarz mit einer Frequenz von 14 MHz. Diese Frequenz ist so hoch, daß man sie auf keinen Fall per Software zählen kann. Auf der GDP-Baugruppe gibt es viele Taktausgänge, die auch eine geteilte Frequenz liefern. Am einfachsten ist die Benutzung des sogenannten VSYNC-Pulses (Vertikal-Synchronisationssignal), der zum Beispiel an der 7-poligen Steckerleiste herausgeführt ist und einen 20ms-Takt liefert. Er wird zum Synchronisieren des Bildes verwendet. Dieser Takt wird vom Graphik-Prozessor selbst aus dem Grundtakt abgeleitet und ist daher quarzgenau. Man kann nun den Takt z.B. über eine IOE-Baugruppe abfragen, wenn man eine Verbindung zu einem entsprechenden Port herstellt. Einfacher aber ist es, dafür gleich den Graphikprozessor selbst zu verwenden. Die Information steht im sogenannten Statusregister auf Adresse $ffffff70.

Dieses Register kann man lesen, so als ob es ein Eingabe-Port auf der IOE-Karte wäre. Dazu kann man einen einfachen Versuch machen. Wir rufen das Menue „IO-Lesen" auf. Dort gibt man die Adresse $ffffff70 an. Als Inhalt der Adresse erscheint am Bit 1, also am 2. Bit von rechts, entweder eine Eins, oder eine Null. Dieses Bit entspricht dem vertikalen Synchronsignal VSYNC. Alle 20 ms geht es für kurze Zeit auf 1. Wenn man nun im Menue „IO-Lesen" die Leertaste drückt, so wird das Register erneut gelesen. Dabei kann das Bit den Wert wechseln, je nachdem, zu welchem Zeitpunkt man die Taste drückt. Wenn man die Taste „D" drückt, so wird der Registerinhalt dauernd gelesen, dann aber erscheint immer eine 1 bei Bit 1, denn das Lesen des Registers wird vom Grundprogramm auch genau alle 20 ms durchgeführt, und zwar genau in der Zeit, wo das Bit 1 auf 1 liegt. Das Grundprogramm macht dies so, weil dadurch ein

Abb. 3.4.21
Erfolgreiche
Landung

```
Rolf-D.Klein 68000/08 Assembler 4.3 (C) 1984, Seite 1


009C00                  ************************************
009C00                  * SIMULATION: DER FLUG EINES BALLES *
009C00                  * ROLF-DIETER KLEIN  850223         *
009C00                  ************************************
009C00
009C00                  * VEREINBARUNGSBAUSTEIN
009C00
009C00                  BALL:
009C00  00 01 02 03 04   DC.B 0,1,2,3,4,5,6,7,10
009C05  05 06 07 0A
009C09  00               DS 0
009C0A
009C0A  0000            X: DC.W 0       * X-KOORDINATE
009C0C  0000            Y: DC.W 0       * Y-KOORDINATE
009C0E  0000            VX: DC.W 0      * GESCHWINDIGKEIT X-RICHTUNG
009C10  0000            VY: DC.W 0      * GESCHWINDIGKEIT Y-RICHTUNG
009C12  0000            T: DC.W 0       * ABGELAUFENE ZEIT, FUER REIBUNG
009C14
009C14                  * PROGRAMMBAUSTEIN
009C14
009C14                  START:                                  zu Abb. 3.4.22
009C14  4279 00009C0A    CLR X          * START BEI X=0
009C1A  33FC 00FA        MOVE #250,Y    * UND Y=250
009C1E  00009C0C
009C22  4279 00009C10    CLR VY         * WIRD DORT FALLENGELASSEN
009C28  33FC 0003        MOVE #3,VX     * ABER MIT ANFANGSGESCHW.
009C2C  00009C0E
```

```
009C30                  SCHLEIFE:
009C30  3239 00009C0A    MOVE X,D1       * BALL DARSTELLEN
009C36  3439 00009C0C    MOVE Y,D2       *
009C3C  303C 0003        MOVE #3,D0      * GROESSE
009C40  41F9 00009C00    LEA BALL,A0     * ADRESSE DES BALLS
009C46  4EB9 0000328C    JSR @FIGUR      * UND DARSTELLEN
009C4C                  WARTE:          * DADURCH ZEITRASTER
009C4C  4EB9 00000E38    JSR @SYNC
009C52  6700 FFF8        BEQ WARTE       *
009C56  0679 0001        ADD #1,T        * ZEIT ERHOEHEN
009C5A  00009C12
009C5E  3039 00009C0E    MOVE VX,D0
009C64  D179 00009C0A    ADD D0,X        * NEUE X-KOORDINATE
009C6A  0479 0001        SUB #1,VY       * GRAVITATION
009C6E  00009C10
009C72  3039 00009C10    MOVE VY,D0
009C78  D179 00009C0C    ADD D0,Y        * NEUE Y-KOORDINATE
009C7E  6A00 FFB0        BPL SCHLEIFE    * FALLS UEBER BODEN, DANN OK
009C82  9179 00009C0C    SUB D0,Y        * SONST REFLEXION
009C88  0679 0001        ADD #1,VY       * VEKTOR GENAU UMGEKEHRT
009C8C  00009C10
009C90  4479 00009C10    NEG VY          * NACH OBEN GERICHTET, PHYS. NICHT GANZ EXAKT
009C96  0479 0001        SUB #1,VY       * ZUSAETZLICH REIBUNG
009C9A  00009C10
009C9E  0C79 0028        CMP #40,T       * ALLE 40*20MS
009CA2  00009C12
009CA6  6F00 FF88        BLE SCHLEIFE
009CAA  4279 00009C12    CLR T           * NEU ZAEHLEN
009CB0  0479 0001        SUB #1,VX
009CB4  00009C0E
009CB8  6A00 FF76        BPL SCHLEIFE
009CBC  4279 00009C0E    CLR VX        * REIBUNG NR IN EINER RICHTUNG WIRKSAM
009CC2  6000 FF6C        BRA SCHLEIFE  * ENDLOS
009CC6
009CC6

008AF0  Ende-Symboltabelle
```

Abb. 3.4.22
Simulation eines Ballfluges

```
sekunde:            * Unterprogramm 1SEK
 move #50-1,d3      * 1 Sek. auszaehlen
warte:              * Durchlauf ist 20ms
 move.b gdp,d0      * Status lesen
 btst #1,d0         * Vertikal Blank pruefen
 beq warte          * warten bis auf Eins
warte1:
 move.b gdp,d0      * wieder Status lesen
 btst #1,d0         * nun warten bis
 bne warte1         * wieder auf Null
 dbra d3,warte      * das Ganze 50 Mal
 rts
```

Abb. 3.5.1
Das Programm wartet eine Sekunde

```
Textstart=009000 Fenster=009079 Ter=00935E      aber CTRL-J=Hilfe
```

flimmerfreies Bild entsteht. Der Bildaufbau erfolgt nämlich immer dann, wenn sich der Schreibstrahl des Bildschirms gerade im unsichtbaren Bereich befindet.

Nun kann man ein Programm schreiben, das dieses Bit abfragt und uns einen 20 ms-Takt liefert. *Abb. 3.5.1* zeigt ein kleines Programmbeispiel. Das Programm „SEKUNDE" wartet genau eine Sekunde und kehrt dann zurück. Dazu wird als erstes im Register der Wert 50-1, also 49, geladen. Damit wird die Schleife zwischen „WARTE" und „DBRA" 50 mal durchlaufen. Man erinnere sich, der DBRA-Befehl zählt immer bis -1.

Bei der Marke „WARTE" wird zunächst das Status-Register des Graphik-Prozessors in das Register D0.B geladen. Mit „BTST" wird anschließend die Lage von Bit 1 geprüft und in das Zero-Flag (Null-Flag) übertragen. Der folgende Sprung „BEQ" springt also solange zur Marke „WARTE" zurück, bis Bit 1 eins geworden ist, also das Vertikale Synchronsignal auftritt. Dann folgt die Marke „WARTE1". Dort wird erneut der Inhalt des Status-Registers gelesen. Nach dem Bit-Test erfolgt diesmal ein Sprung, wenn das Bit ungleich Null ist, zurück zur Marke „WARTE1". Diese zweite Schleife ist nötig, um sicherzustellen, daß das Programm beim DBRA-Befehl wirklich nur alle 20 ms ankommt. Was würde passieren, wenn man die zweite Schleife weglassen würde? Dann würde die Schleife „WARTE" verlassen werden, sobald das Bit 1 des Statusregisters auf 1 geht, und das Programm würde zum DBRA-Befehl kommen. Damit käme das Programm sofort wieder zur Marke „WARTE", noch während Bit 1 auf eins steht. Die Schleife würde also sofort durchlaufen, und ein neuer DBRA-Befehl würde folgen. Darum ist es nötig, nach dem Auftreten des Synchronsignals in einer zweiten Schleife zu warten, bis es wieder verschwunden ist. Nur dadurch ist sichergestellt, daß jedes Auftreten des Synchronsignals auch nur einmal gezählt wird.

Im Grundprogramm befindet sich eine Routine, die man dazu direkt verwenden kann. *Abb. 3.5.2* zeigt das Programm nochmals unter Verwendung des Unterprogramms „SYNC". Das Unterprogramm liefert einen Wert ungleich null, sobald das Vertikal-Synchronsignal aufgetreten ist, sonst null. Dabei wird auch berücksichtigt, daß der Wert wirklich nur alle 20 ms auftritt. Wenn man das Programm „SEKUNDE" aufruft, so vergeht eine Sekunde, bis es wieder zurückkehrt. Wird es über das Start-Menue direkt gestartet, bleibt der Bildschirm eine Sekunde lang dunkel, und dann meldet sich das Grundprogramm wieder mit „M=Menue F=Flip".

Wir wollen zuerst ein kleines Programm schreiben, das uns eine Zeit auf dem Bildschirm anzeigt. *Abb. 3.5.3* zeigt das Struktogramm. Die Zeit soll dauernd ausgegeben werden, daher wird eine Schleife, die unbegrenzt abläuft, verwendet. Dann wird auf den 20ms-Puls gewartet. Danach wird ein entsprechender Zeitzähler erhöht, worauf die Zeit auf dem Bildschirm ausgegeben wird.

Abb. 3.5.4 zeigt die Programmrealisierung. Als Zählregister wird D4.L verwendet. Es wird mit dem CLR- (Clear oder Löschen) Befehl auf 0 gesetzt. Dann wird die Warteschleife ausgeführt. Erst wenn SYNC einen Wert ungleich 0 liefert, ist der 20ms-Puls am VSYNC-Ausgang angekommen. Danach wird der Wert 20 zum Register D4.L addiert. Nun folgt die Ausgabe des Wertes. Dazu wird zunächst die Adresse BUFFER in das Adreßregister A0 geladen. Dorthin soll der Ausgabetext geschrieben werden. Der Wert D4.L wird in das Register D0.L geholt. Nun wird mit DIVU (divide unsigned oder dividiere ohne Vorzeichen) der Wert durch 100 geteilt, um eine Ausgabe in 1/10

```
sekunde:          * Unterprogramm 1SEK
 move #50-1,d3    * 1 Sek. auszaehlen
warte:            * Durchlauf ist 20ms
 jsr @sync        * Unterprog, liefert
 beq warte        * alle 20ms Wert <>0
 dbra d3,warte    * das Ganze 50 Mal
 rts
```

```
Textstart=009000 Fenster=009079 Tor=00935E      amer CTRL-J=Hilfe
```

Abb. 3.5.2
Durch Verwendung eines Unterprogramms aus dem Grundprogramm wird das Programm kürzer

Abb. 3.5.3
Das Struktogramm zum Programm: Zeitausgabe

```
**********************
* Uhr fuer Zeitnahme *
**********************

zeit:             * Uhrenprogramm-Start
 clr d4.l         * Zeitzaehler loeschen
schleife:         * Wiederhole
 jsr @sync        *   warten bis 20 ms
 beq schleife     *  dann weiter
 add.l #20,d4     *  dort ist der ms-Zaehler
 lea buffer,a0    *  Ziel fuer Textausgabe
 move.l d4,d0     *  D0 ist Parameter
 divu #100,d0     *  Ausgabe in 1/10 Sek.
 jsr @print4d     *  Umwandlung in ASCII
 lea buffer,a0    *  fuer Ausgabe vorbereiter
 move #$22,d0     *  Schriftgroesse 2,2
 move #0,d1       *  x=0
 move #220,d2     *  y=220
 jsr @write       *  und Text ausgeben
 bra schleife     * unbegrenzt wiederholen

buffer: ds.b 80 * Speicherbereich fuer Text
```

```
Textstart=009000 Fenster=009000 Tor=009000 einf amer CTRL-J=Hilfe
```

Abb. 3.5.4
Ein einfaches Uhrenprogramm

Sekunden zu erreichen. Die Umrechnung in einen ASCII-Text erfolgt mit dem Unterprogramm PRINT4D, das einen Wert im Register D0.W als 16-Bit-Zahl interpretiert und als dezimale Ziffernfolge in ASCII, beginnend bei der Adresse in A0 aufsteigend ablegt. Das Ende wird durch Ablage des Codes 0 gekennzeichnet, so wie es der WRITE-Befehl anschließend braucht. Natürlich ist bei der Ausgabe von längeren Zahlen darauf zu achten, daß beim folgenden Umrechnen von kürzeren Zahlen die hohen Stellen wieder auf null gesetzt werden. Bei uns ist das kein Problem, denn die Zahl wird ja immer größer (bis zu 65535, dann allerdings gibt's Probleme). Die Ausgabe des Buffers erfolgt mit WRITE. Danach springen wir zurück zum Schleifenanfang „SCHLEIFE".

Nach dem Start des Programms sieht man die Zeitangabe in Zehntelsekunden eingeblendet.

Eine Standuhr

Mit dem Unterprogramm SYNC kann man schon eine ganze Menge anfangen. Aufgabe ist es, eine Standuhr mit Pendel und Ziffernblatt auf dem Bildschirm darzustellen. Der Zeitgeber soll das Unterprogramm SYNC sein.

Abb. 3.5.5 zeigt die komplette Programmlösung. Entgegen unseren bisherigen Gewohnheiten geben wir das Programm beginnend bei der Adresse $A000 ein. Dazu muß man das Optionen-Menue anwählen und dort mit „Textstart" die Adresse $A000 vorwählen. Dann beantwortet man die Frage „wirklich ?" durch Eingabe von „J" und geht zurück zum Editor. Dort muß nun links unten „Textstart = 0000A000" stehen. Dann kann man das Programm eingeben. Der Objekt-Code, also das Maschinenprogramm, wird hier bei Adresse $9C00 abgelegt und reicht bis Adresse $9EE8.

Wenn man das Programm mal ergänzen will, so muß man ggf. mit einer „ORG"-Anweisung den Maschinencode z. B. bei Adresse $9000 ablegen.

Doch nun zum Programm. Im Vereinbarungsbaustein werden neben der Konstanten „GDP" auch einige Variable festgelegt. Das Pendel soll in verschiedenen Stellungen auf dem Bildschirm sichtbar sein. *Abb. 3.5.6* zeigt die Standuhr mit einer dieser Pendelstellungen. Das Programm soll 10 Stellungen unterscheiden. Ferner soll es den Zeiger auf dem Ziffernblatt in unterschiedlichen Stellungen darstellen.

Für die Zeigerstellungen sind die Variablen „MINWINKEL" und „STDWINKEL" verantwortlich. Sie beinhalten die Winkelstellung des Zeigers. Dabei reicht der Winkel von 0 bis 359°. Der Winkel beträgt 0°, wenn der Zeiger nach oben zeigt, und 90°, wenn er nach rechts gerichtet ist. Es wird also der Winkel im Uhrzeigersinn und nicht im mathematischen Sinn gerechnet. Zwei Zähler „MINZAEHLER" und „STDZAEHLER" beinhalten Werte, die zum Abzählen der Uhrzeit verwendet werden. Da der Minutenzeiger in Grad-Schritten bewegt werden soll, entspricht 1 Grad einer Zeitspanne von 10 Sekunden. Denn 360 Grad in 60 Teile geteilt ergibt 6, also 6° pro Minute, oder 10 Sekunden pro Grad.

Beim Stundenzeiger ist es ähnlich. Hier werden 360° in 12 Stunden geteilt, also 30° pro Stunde. Damit enspricht 1° aber einer Zeitspanne von 2 Minuten oder 120 Sekunden. Daher wird in „MINZAEHLER" ein Wertebereich von 0 bis 9 zugelassen und in

```
009C00                  ****************************
009C00                  * STANDUHRPROGRAMM   1.0   *
009C00                  ****************************
009C00                  * AUF ADRESSE $A000 IM EDITOR EINGEBEN
009C00                  * CODE AB $9C00
009C00
009C00                  * VEREINBARUNGS-BAUSTEIN
009C00
= FFFFFF70              GDP EQU $FFFFFF70     * STATUSPORT GDP
009C00
009C00  0000            PENDELPOS: DC.W 0     * PENDELPOSITION  0,2,4,6,...18 ALS INDEX

009C02  0002            RICHTUNG:  DC.W 2     * PENDELRICHTUNG  UND GESCHWINDIGKEIT
009C04  0000            MINWINKEL: DC.W 0     * 0..359 GRAD   6 GRAD PRO MINUTE
009C06  005A            STDWINKEL: DC.W 90    * 0..359 GRAD  30 GRAD PRO STUNDE
009C08  0000            MINZAEHLER: DC.W 0    * 0..9
009C0A  0000            STDZAEHLER: DC.W 0    * 0..119
009C0C
009C0C                  PENTAB:
009C0C  FFF6 FFF7 FFF8   DC.W -10,-9,-8,-5,-2,2,5,8,9,10
009C12  FFFB FFFE 0002
009C18  0005 0008 0009
009C1E  000A
009C20
009C20                  * UNTERPROGRAMM-BAUSTEIN
009C20
009C20                  SEK10TEL:       * UNTERPROGRAMM 1SEK
009C20  363C 0004        MOVE #5-1,D3   * 1/10 SEK. AUSZAEHLEN
009C24                  WARTE:          * DURCHLAUF IST 20MS
009C24  4EB9 00000E38    JSR @SYNC      * UNTERPROG, LIEFERT
009C2A  6700 FFF8        BEQ WARTE      * ALLE 20MS WERT <>0
009C2E  51CB FFF4        DBRA D3,WARTE  * DAS GANZE 50 MAL
009C32  4E75             RTS
009C34
009C34                  ZAEHLE:         * NEUER SEKUNDENTAKT
009C34  4EB9 00000D6E    JSR @ERAPEN    * ALLE ZAEHLER AUF NEUEN STAND
009C3A  6100 0122        BSR ZEIGERAUS  * ABSCHALTEN DER ZEIGER
009C3E  0679 0001        ADD #1,MINZAEHLER      * ZAEHLT ALLE SEKUNDE
009C42  00009C08
009C46  0C79 000A        CMP #10,MINZAEHLER     * VON 0 BIS 9
009C4A  00009C08
009C4E  6D00 0022        BLT ZAE1
009C52  4279 00009C08    CLR MINZAEHLER
009C58  0679 0001        ADD #1,MINWINKEL       * 1 GRAD ENTSPRICHT 10 SEKUNDEN
009C5C  00009C04
009C60  0C79 0168        CMP #360,MINWINKEL
009C64  00009C04
009C68  6D00 0008        BLT ZAE1
009C6C  4279 00009C04    CLR MINWINKEL
009C72                  ZAE1:
009C72  0679 0001        ADD #1,STDZAEHLER      * ZAEHLT SEKUNDEN, VON 0..119
009C76  00009C0A
009C7A  0C79 0078        CMP #120,STDZAEHLER
009C7E  00009C0A
009C82  6D00 0022        BLT ZAE2
```

zu Abb. 3.5.5

```
009C86  4279 00009C0A    CLR STDZAEHLER
009C8C  0679 0001        ADD #1,STDWINKEL       * 1 GRAD ENTPRICHT 120 SEKUNDEN
009C90  00009C06
009C94  0C79 0168        CMP #360,STDWINKEL     *
009C98  00009C06
009C9C  6D00 0008        BLT ZAE2
009CA0  4279 00009C06    CLR STDWINKEL
009CA6                  ZAE2:
009CA6  4EB9 00000D7C    JSR @SETPEN            * NEUE ZEIGERSTELLUNG AUSGEBEN
009CAC  6100 00B0        BSR ZEIGERAUS
009CB0  4E75             RTS
009CB2
009CB2
009CB2                  PENDEL:          * MIT SCHILDKROETEN-
009CB2  303C 0064        MOVE #100,D0    * GRAPHIK, PENDEL DARSTELLEN
009CB6  4EB9 00001288    JSR @SCHREITE
009CBC  303C FFA6        MOVE #-90,D0    * PERPENDIKEL ORIENTIEREN
009CC0  4EB9 000012F6    JSR @DREHE
009CC6  363C 0009        MOVE #10-1,D3
009CCA                  PENWDH:
009CCA  303C 0003        MOVE #3,D0
009CCE  4EB9 00001288    JSR @SCHREITE
009CD4  303C 0024        MOVE #36,D0
009CD8  4EB9 000012F6    JSR @DREHE
009CDE  51CB FFEA        DBRA D3,PENWDH
009CE2  4E75             RTS
009CE4
009CE4                  PENDELAUS:       * AUF AKTUELLER POSITION AUSGEBEN
009CE4  323C 0100        MOVE #256,D1    * JE NACH WINKEL
009CE8  343C 0100        MOVE #256,D2
009CEC  41F9 00009C0C    LEA PENTAB,A0
009CF2  3639 00009C00    MOVE PENDELPOS,D3
009CF8  D0C3             ADDA.W D3,A0
009CFA  363C 010E        MOVE #270,D3          * MITTELPOSITION BEI 9
009CFE  D650             ADD (A0),D3
009D00  4EB9 0000131A    JSR @SET
009D06  4EB9 00001112    JSR @HIDE       * SCHILDKROETE SELBST AUSBLENDEN
009D0C  6100 FFA4        BSR PENDEL
009D10  4E75             RTS
009D12
009D12                  ZAEPENDEL:       * ALLE 1/10 SEK NEUE PENDELPOSITION
009D12  4EB9 00000D6E    JSR @ERAPEN
009D18  6100 FFCA        BSR PENDELAUS   * ALTES LOESCHEN
009D1C  3039 00009C00    MOVE PENDELPOS,D0
009D22  D079 00009C02    ADD RICHTUNG,D0
009D28  33C0 00009C00    MOVE D0,PENDELPOS
009D2E  0C79 0012        CMP #18,PENDELPOS
009D32  00009C00
009D36  6D00 0008        BLT ZAP1        * 0,2,4,6,...18 ERLAUBT
009D3A  4479 00009C02    NEG RICHTUNG    * UMDREHEN DES RICHTUNGSSINNS
009D40                  ZAP1:
009D40  0C79 0000        CMP #0,PENDELPOS
009D44  00009C00                                              zu Abb. 3.5.5
009D48  6E00 0008        BGT ZAP2        * 0 IST MINIMUM
009D4C  4479 00009C02    NEG RICHTUNG    * AUS -2 WIRD 2
009D52                  ZAP2:
009D52  4EB9 00000D7C    JSR @SETPEN
009D58  6100 FF8A        BSR PENDELAUS * PENDEL WIEDER DARSTELLEN
009D5C  4E75             RTS
```

```
009D5E
009D5E
009D5E                    ZEIGERAUS:  * ANHAND DER WINKEL, ZEIGER DARSTELLEN
009D5E  323C 0100          MOVE #256,D1
009D62  343C 0128          MOVE #256+40,D2
009D66  363C 005A          MOVE #90,D3
009D6A  9679 00009C04      SUB MINWINKEL,D3
009D70  4EB9 0000131A      JSR @SET
009D76  4EB9 00001112      JSR @HIDE
009D7C  303C 0016          MOVE #22,D0 * MINUTENZEIGER
009D80  4EB9 00001288      JSR @SCHREITE
009D86  323C 0100          MOVE #256,D1
009D8A  343C 0128          MOVE #256+40,D2
009D8E  363C 005A          MOVE #90,D3
009D92  9679 00009C06      SUB STDWINKEL,D3
009D98  4EB9 0000131A      JSR @SET
009D9E  4EB9 00001112      JSR @HIDE
009DA4  303C 0010          MOVE #16,D0 * STUNDENZEIGER
009DA8  4EB9 00001288      JSR @SCHREITE
009DAE  4E75               RTS
009DB0
009DB0                    ZIFFERBLATT:     * ZIFFERNBLATT DARSTELLEN
009DB0  323C 010Q          MOVE #256,D1
009DB4  343C 010A          MOVE #256+10,D2
009DB8  363C 0000          MOVE #0,D3
009DBC  4EB9 0000131A      JSR @SET
009DC2  363C 000B          MOVE #12-1,D3
009DC6                    ZIFWDH:
009DC6  303C 005A          MOVE #90,D0
009DCA  4EB9 000012F6      JSR @DREHE
009DD0  303C 0005          MOVE #5,D0
009DD4  4EB9 00001288      JSR @SCHREITE
009DDA  303C FFFB          MOVE #-5,D0
009DDE  4EB9 00001288      JSR @SCHREITE
009DE4  303C FFA6          MOVE #-90,D0
009DE8  4EB9 000012F6      JSR @DREHE
009DEE  303C 000F          MOVE #15,D0
009DF2  4EB9 000012F6      JSR @DREHE
009DF8  303C 00FA          MOVE #250,D0
009DFC  4EB9 0000128A      JSR @SCHR16TEL
009E02  303C 000F          MOVE #15,D0
009E06  4EB9 000012F6      JSR @DREHE
009E0C  51CB FFB8          DBRA D3,ZIFWDH
009E10  4E75               RTS
009E12
009E12                    GEHAEUSE:        * UHRENGEHAEUSE DARSTELLEN
009E12  303C 005A          MOVE #90,D0
009E16  4EB9 000012F6      JSR @DREHE
009E1C  303C FFE2          MOVE #-30,D0
009E20  4EB9 00001288      JSR @SCHREITE
009E26  303C 003C          MOVE #60,D0
009E2A  4EB9 00001288      JSR @SCHREITE
009E30  303C 005A          MOVE #90,D0
009E34  4EB9 000012F6      JSR @DREHE
009E3A  303C 0096          MOVE #150,D0
```

zu Abb. 3.5.5

```
Rolf-D.Klein 68000/08 Assembler 4.3 (C) 1984, Seite 4

009E3E  4EB9 00001288     JSR @SCHREITE
009E44  303C 005A         MOVE #90,D0
009E48  4EB9 000012F6     JSR @DREHE
009E4E  303C 003C         MOVE #60,D0
009E52  4EB9 00001288     JSR @SCHREITE
009E58  303C 005A         MOVE #90,D0
009E5C  4EB9 000012F6     JSR @DREHE
009E62  303C 00A0         MOVE #150+10,D0
009E66  4EB9 00001288     JSR @SCHREITE
009E6C  303C 005A         MOVE #90,D0
009E70  4EB9 000012F6     JSR @DREHE
009E76  303C 003C         MOVE #60,D0
009E7A  4EB9 00001288     JSR @SCHREITE
009E80  303C FFA6         MOVE #-90,D0
009E84  4EB9 000012F6     JSR @DREHE
009E8A  303C FFF6         MOVE #-10,D0
009E8E  4EB9 00001288     JSR @SCHREITE
009E94  303C 0046         MOVE #70,D0
009E98  4EB9 00001288     JSR @SCHREITE
009E9E  303C FFA6         MOVE #-90,D0
009EA2  4EB9 000012F6     JSR @DREHE
009EA8  303C 003C         MOVE #60,D0
009EAC  4EB9 00001288     JSR @SCHREITE
009EB2  303C FFA6         MOVE #-90,D0
009EB6  4EB9 000012F6     JSR @DREHE
009EBC  303C 003C         MOVE #60,D0
009EC0  4EB9 00001288     JSR @SCHREITE
009EC6  4E75              RTS
009EC8
009EC8                    * PROGRAMMBAUSTEIN
009EC8
009EC8                    START:            * HAUPTPROGRAMM
009EC8  6100 FF48          BSR GEHAEUSE     * RAHMEN ZEICHNEN
009ECC  6100 FEE2          BSR ZIFFERBLATT  * UND ZIFFERNBLATT AUSGEBEN
009ED0                    WIEDERHOLE:
009ED0  383C 0009          MOVE #10-1,D4    * 10 MAL 1/10 SEKUNDE
009ED4                    WDH1:
009ED4  6100 FD4A          BSR SEK10TEL     * 1/10 SEK VERZOEGERN
009ED8  6100 FE38          BSR ZAEPENDEL    * ZAEHLEN UND
009EDC  51CC FFF6          DBRA D4,WDH1     * PENDEL DARSTELLEN
009EE0  6100 FD52          BSR ZAEHLE       * DANN ZEIGER DARSTELLEN
009EE4  6000 FFEA          BRA WIEDERHOLE   * UNBEGRENZT
009EE8
009EE8                     END

008CB2  Ende-Symboltabelle
```

Abb. 3.5.5 Das Programm „Standuhr“

Abb. 3.5.6
So sieht die Standuhr
auf dem Bildschirm aus

„STDZAEHLER" ein Bereich von 0 bis 119. Beide Zähler werden nach dem Ablauf einer Sekunde erhöht.

Die Pendelposition wird aus dem Inhalt des Zählers „PENDELPOS" abgeleitet. Er zählt in Zweierschritten von 0 bis 18. Enthält er den Wert 0, so soll das Pendel nach links geschwenkt sein, enthält er 18, so ist die rechte Endposition einzunehmen. Der Zählerstand wird dabei als Index für die Tabelle „PENTAB" verwendet, die die entsprechenden Winkel des Pendels enthält. Dabei wird auch berücksichtigt, daß das Pendel beim Hin- und Herschwingen im Nulldurchgang schneller ist als am Rand, wo es ja kurzzeitig sogar Stillstand erlangt. Ist „PENDELPOS" auf 0, so wird der erste Wert in der Tabelle verwendet, also -10. Bei einem Wert von 2, wird der Wert -9 verwendet, dann bei 4 der Wert -8 usw. „PENDELPOS" schreitet in Zweierschritten, da ein Eintrag in „PENTAB" immer ein Wort ist, also zwei Bytes belegt.

Nun folgen die Unterprogramm-Bausteine. Das Unterprogramm „SEK10TEL" wartet 1/10 Sekunde, dann kehrt es zurück. Als Zeitgeber wird dabei der SYNC-Aufruf verwendet. Das Unterprogramm „ZAEHLE" löscht zuerst die alten Zeiger, berechnet dann die neuen Winkel und blendet schließlich die neue Zeigerstellung ein.

Das Unterprogramm PENDEL zeichnet unter Verwendung der Schildkrötengraphik das Pendel; es wird vom Unterprogramm PENDELAUS aufgerufen. PENDELAUS lädt zunächst die Adresse der Tabelle PENTAB in das Register A0, addiert den Index, also „PENDELPOS" darauf und lädt dann 270 in das Register D3. Dies ist die Ruheposition des Pendels. Dazu wird der aktuelle Winkelwert addiert. Das Pendel wird anschließend mit diesem neuen Winkelwert, durch den Aufruf von PENDEL, ausgegeben. Die Mittelposition wird nie exakt erreicht; sie läge bei einem Index von 9, der aber nicht zulässig ist.

Bei „ZAEPENDEL" wird der Index ermittelt. Am Anfang ist der Inhalt der Variablen „RICHTUNG" gleich 2. Damit wird immer 2 auf den Inhalt von „PENDELPOS" addiert. Der Inhalt von „PENDELPOS" nimmt somit die Werte von 0,2,4,6,8,10,12,14,16 bis 18 an. Wenn der Wert 18 erreicht ist, wird die Variable „RICHTUNG" negiert, also der Wert -2 nach „RICHTUNG" gespeichert. Beim nächsten Durchlauf wird dann wieder der Wert 16,14,... bis 0 in „PENDELPOS" verwendet. Wenn 0 auftritt, wird „RICHTUNG" wiederum negiert und somit wieder der Wert 2 verwendet, das Pendel bewegt sich wieder in ursprünglicher Richtung, usw. Dadurch bewegt sich das Pendel hin und her.

In „ZEIGERAUS" wird anhand der Winkelangaben in „MINWINKEL" und „STDWINKEL" die Zeigerstellung ausgegeben. Das Unterprogramm „ZIFFERBLATT" stellt das Zifferblatt dar, mit „GEHAEUSE" wird die Standuhr ausgegeben.

Damit wären die einzelnen Unterprogramme klar; wir können uns dem Hauptprogramm START widmen. Zunächst wird das Gehäuse ausgegeben und das Zifferblatt eingeblendet. Dann beginnt die Schleife „WIEDERHOLE". Die Schleife „SEK10TEL" und „ZAEPENDEL" wird 10 mal durchlaufen und dann mit „ZAEHLE" eine neue Darstellung der Ziffern durchgeführt. „ZAEHLE" wird damit jede Sekunde einmal aufgerufen.

Leider hat unser Programm noch einen Schönheitsfehler. Vielleicht wird sich bei Ihnen nach mehreren Stunden eine Abweichung der Uhrzeit ergeben, denn die Unterprogramme müßten, um eine Zeitverfälschung auszuschließen, immer weniger als 20

Millisekunden dauern. Dies ist dann besonders kritisch, wenn sowohl das Pendel als auch die Ziffern neu dargestellt werden.

Dann wird ein „SYNC"-Befehl verschluckt, und es gehen 20 Millisekunden verloren. Um das zu verhindern, gibt es mehrere Möglichkeiten. Man kann mehrere „SYNC"-Befehle in Programmteile verstreuen und muß dort beim Auftreten eines SYNCs einen Hilfszähler um 20 Millisekunden erhöhen. Bei Ablauf einer Sekunde kann man dann die anderen Zeiten einfach korrigieren. Diese Methode ist aber nicht ganz einfach und auch nicht sicher. Besser ist es, einen anderen Mechanismus bei der Zeitermittlung zu verwenden, den wir jetzt gleich kennenlernen werden, und der auch noch andere interessante Eigenschaften besitzt.

Zeiteinblendung bei der Mondlandung

Bei der Mondlandung ist das Problem ähnlich. Wie kann man eine genaue Zeit ermitteln, ohne das Programm MONDLANDUNG völlig neu konstruieren zu müssen. Es sollte möglich sein, eine Zeitermittlung zu finden, die keine Rücksicht auf die Zeitdauer von Bildschirmausgaben nehmen muß. Man verwendet Interrupts.

Die Interrupttechnik

Interrupt bedeutet wörtlich Unterbrechung, man kann also den Lauf eines Programms unterbrechen. Dabei wird die Unterbrechung des Programms durch ein elektrisches Signal ausgelöst. Wenn man die RESET-Taste drückt wird z. B. der Programmablauf unterbrochen, da gibt es aber keine Möglichkeit mehr das Programm, das unterbrochen wurde anschließend weiter zu führen. Bei einem Interrupt ist es anders. Das Interrupt-Programm wird aufgerufen, wie ein Unterprogramm. Nach Auslösung des Interruptprogramms wird die Rückkehradresse gespeichert und man kann später das unterbrochene Programm wieder fortsetzen.

Der Assemblerbefehl RTE

Da der Interrupt über eine Leitung mit einem elektrischen Signal ausgelöst wird, muß die CPU auch entsprechende Eingänge besitzen. Beim 68008 sind es zwei, beim 68000 gar drei Interrupteingänge, die unterschiedliche Eigenschaften besitzen. Die Leitungen sind beim 68008 mit IPL0/2-Quer und mit IPL1-Quer beschriftet. Beim 68000 sind die Leitungen IPL0-Quer und IPL2-Quer getrennt verfügbar. Wenn man die Leitungen IPL0/2-Quer und IPL1-Quer zusammen auf 0 V legt, so wird ein Interrupt ausgelöst. Das laufende Programm wird unterbrochen, und der Prozessor springt auf eine spezielle Adresse. Dort steht dann das sogenannte Interruptprogramm, das abgearbeitet wird. Am Schluß des Interruptprogramms steht ein Rücksprungbefehl, ähnlich wie bei einem Unterprogramm. Dieser Befehl heißt: RTE oder return from exception, Rückkehr von der

Ausnahmebehandlung. Nach dieser Rückkehr wird das unterbrochene Programm wieder weiter bearbeitet. Das Interruptprogramm kann in unserem Falle den Zeitzähler bei jedem Interrupt um einen Wert erhöhen. Der Interrupt kann zum Beispiel direkt durch die Leitung VSYNC ausgelöst werden. Im Hauptprogramm kann man dann den Zeitzähler nach jedem Spielblock-Durchlauf ausgeben und ist unabhängig von der Dauer eines Spieldurchlaufs. Interruptprogramme sind also praktisch Unterprogramme. Dabei muß man aber einiges besonders beachten. Der Interrupt kann an jeder beliebigen Stelle des normalen Programmablaufs auftreten. Das Programm wird dann sofort unterbrochen. Es können also, was normalerweise der Fall sein wird, gerade wichtige Werte für unser Hauptprogramm in den Registern stehen. Wenn man nun einfach beliebige Register im Interruptprogramm für eigene Zwecke verwendet, so kann es sein, daß man damit diese Daten des Hauptprogramms zerstört. Wie kann man das verhindern? Zum einen, indem man im Interruptprogramm einfach Register verwendet, die nicht im Hauptprogramm (in allen Programmteilen des Hauptprogramms) verwendet werden. Nun das, ist bei uns nicht möglich, denn auch Unterprogramme, die im Hauptprogramm aufgerufen werden, können Register verwenden. Und da auch Programmteile aus dem Grundprogramm aufgerufen werden, von denen man nicht immer weiß, welche Register sie brauchen, scheidet diese Möglichkeit aus. Man muß also einen Weg finden, die Registerinhalte zu retten. Dazu könnte man sie einfach in Variablenplätze abspeichern, und vor dem Verlassen des Interruptprogramms lädt man sie wieder zurück. Besser und einfacher ist die Verwendung des sogenannten Stacks. Stack bedeutet Stapelspeicher. Und wie bei jedem Stapel werden Elemente, die man zuletzt abgelegt hat, zuerst wieder geholt. Dazu braucht man einen Stackpointer. Wir verwenden das Register A7, ein Adreßregister, das dazu die Adresse der aktuellen Stapeloberkante enthält. Das Adreßregister ist schon mit einem sinnvollen Wert vorbelegt, da es auch vom Grundprogramm als Stackpointer benötigt wird. *Abb. 3.5.7* zeigt das Schema.

Der Assemblerbefehl MOVE.L D0,-(A7)

Man kann jedes Adreßregister als Stackpointer verwenden, jedoch ist A7 dafür besonders geeignet, da es beim 68000/8 einige Befehle gibt, die automatisch dieses Register verwenden. *Abb. 3.5.8* zeigt nun eine Registerbelegung. Das Register A7 soll die Adresse $9FFC gespeichert haben. Damit zeigt es auf eine Speicherzelle. Mit dem Befehl MOVE.L (A7),D0 könnte man z. B. ein Langwort von dieser Adresse nach D0 laden. Es gibt aber für unser Vorhaben einen speziellen Befehl, der besser geeignet ist: MOVE.L D0,-(A7). Im Register D0 steht z.B. der Wert 4. Nach Ausführung des MOVE-Befehls ergibt sich *Abb. 3.5.9*, A7 wurde automatisch erniedrigt, unser Stapel wächst also gewissermaßen rückwärts. Es enthält dann den Wert $9FF8 (die neue Stapeloberkante), und dort steht auch der Wert 4.

Das Minus vor der Klammer bei dem MOVE-Befehl bedeutet, daß vor Ausführung des Transports zunächst das Adreßregister A7 verringert wird, und zwar um den Wert der Wortbreite, also bei einem Langworttransport um den Wert 4. Danach wird der Inhalt des Registers A7 als Adresse aufgefaßt und dorthin der Inhalt des Registers D4 geladen.

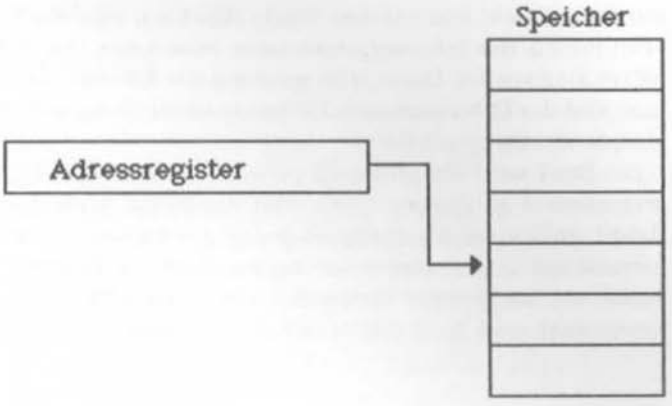


Abb. 3.5.7
Der Stackpointer

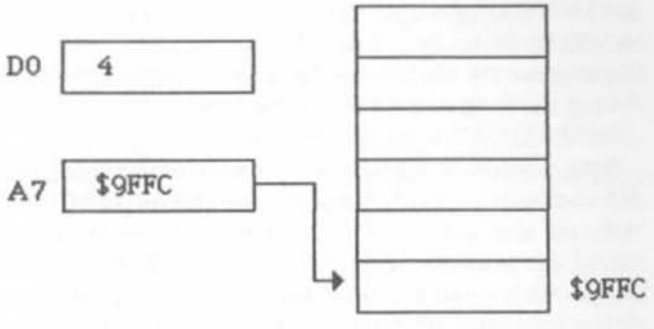


Abb. 3.5.8
Register-Belegung
bei Start

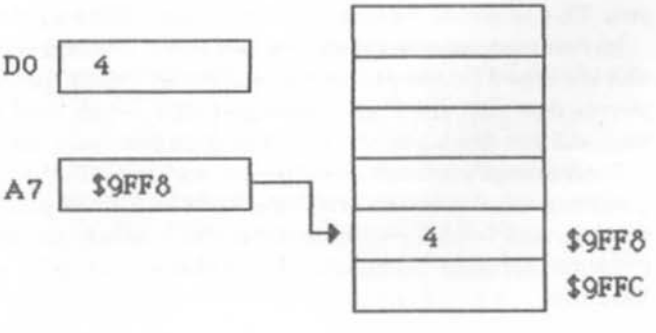


Abb. 3.5.9
Nach Ausführung
des MOVE-Befehls

Der Wert 4 steht nun auf dem Stack. Nun kann man das Register D4 für seine Arbeit, bzw. für die des Interruptprogrammes verwenden. Am Schluß gibt man den Befehl: MOVE.L (A7)+,D4. Damit wird zunächst die Adresse von A7 als Quelladresse genommen und der Datenwert nach D4 transportiert. Dann wird A7 wieder um die entsprechende Wortlänge erhöht.

Der Stack wird übrigens auch verwendet, um die Rückkehradressen beim Unterprogrammaufruf zu merken. Dazu wird die Rückkehradresse bei einem BSR oder JSR-Befehl einfach auf den Stack gelegt und das Register A7 um 4 veringert. Das geschieht automatisch und immer unter Verwendung des Registers A7. Daher sollte man A7 immer als Stackpointer verwenden. Bei einem RTS-Befehl wird die Rückkehradresse automatisch vom Stack geholt und der Programmzähler mit diesem Langwort geladen.

Der Assemblerbefehl MOVEM.L ...

Abb. 3.5.10 zeigt ein kurzes Programmstück für die Anwendung des Rettebefehls. Man kann den Rettebefehl also auch ganz gut beim normalen Programmieren verwenden, z. B. wenn einem die Register ausgehen und man durch das Retten der alten, aber noch wichtigen Werte Platz schaffen will. Wenn man mehrere Register retten will, so gibt es den MOVEM-Befehl, bei dem man mehrere Register angeben kann. Beispiel:
MOVEM.L D0-D4/D6/A2-A4,-(A7)
Die Register D0 bis D4 und D6 sowie A2 bis A4 werden auf den Stack gerettet. Am Schluß des Programmteils schreibt man:

MOVEM.L (A7)+,D0-D4/D6/A2-A4

Dann werden die Register wieder vom Stack zurückgeladen. Man muß darauf achten, daß man beim Zurückladen genau die gleiche Anzahl von Registern angibt, denn sonst bleibt ein Wert auf dem Stack, oder es wird einer zuviel geholt, und beim nächsten RTS-Befehl, der ja an der Stapeloberkante seine Rücksprungadresse erwartet, geschieht ein Chaos. Auch wenn man den normalen MOVE-Befehl verwendet, muß man aufpassen, daß man Register, die man zuerst auf den Stack legt, zuletzt wieder herunterholt. Beim MOVEM-Befehl wird dies automatisch beachtet.

Man kann auch einzelne Wörter auf dem Stack speichern (MOVE.W oder MOVEM.W) und auch Bytes. Verwendet man dafür das Register A7, so wird dies automatisch immer in 2er-Schritten verändert. Bei der Verwendung anderer Adreßregister ist dies nicht der Fall. Um hier nicht den Überblick zu verlieren, sollte man mit Byte-Ablagen vorsichtig sein. Eine ungerade Adresse im Adreßregister führt zu einem ADDRESS ERROR.

Im Interruptprogramm speichert man zuerst alle Register ab, die man dort verwendet, und vor dem RTE-Befehl lädt man sie wieder zurück. Dann gibt es noch das Statusregister, in dem sind alle Flags gespeichert sind. Dieses wird aber vom Prozessor automatisch auf den Stack gerettet, und man muß sich nicht weiter darum kümmern.

Der Interrupt gehört zu den sogenannten EXCEPTIONs, also zu den Ausnahmebedingungen. Auch der bereits erwähnte ADRESS ERROR gehört dazu. Diese EXCEPTIONs werden vom Grundprogramm automatisch behandelt, eine entsprechende Meldung erscheint auf dem Bildschirm. Die *Tabelle 3.5.1* zeigt eine Zusammenstellung der Vektoren.

Ausnahmevektoren beim 68000 (68008):

Vektor nummer	Adr.	Bedeutung
0	0000	Reset: Stackpointer
	0004	Reset: Programmzähler
2	0008	Bus Error (wenn BERR)
3	000C	Address Error (ungerade Adresse)
4	0010	Illegal Instruction (Befehl unbekannt)
5	0014	Zero Divide (Division durch 0)
6	0018	CHK-Instruktion (Bereichsgrenzen)
7	001C	TRAPV-Inst. (überlauf bei TRAPV)
8	0020	Privilege Violation (priviligierter Befehl)
9	0024	Trace (Trace ist aktiv, nach Befehl)
10	0028	LINE 1010 Emulator
11	002C	LINE 1111 Emulator
12	0030	für Erweiterungen reserviert
13	0034	für Erweiterungen reserviert
14	0038	für Erweiterungen reserviert
15	003C	Uninitialized Interrupt Vektor
16-23	0040	für Erweiterungen reserviert
	005F	-
24	0060	Spurious Interrupt (bei BERR + INT)
25	0064	Level 1 Interrupt Autovektor (68000)
26	0068	Level 2 Interrupt Autovektor (68008) NMI
27	006C	Level 3 Interrupt Autovektor (68000)
28	0070	Level 4 Interrupt Autovektor (68000)
29	0074	Level 5 Interrupt Autovektor (68008) INT
30	0078	Level 6 Interrupt Autovektor (68000)
31	007C	Level 7 Interrupt Autovektor (68008) NMI+INT
32-47	0080	Trap-Instruktion Vektors
	00BF	-
48-63	00C0	für Erweiterungen Reserviert
	00FF	-
65-256	0100	User Interrupt Vektoren
	03FF	-

Tabelle 3.5.1 Ausnahmevektoren beim 68000 und 68008

```
*
* Beispiel fuer die Verwendung eines
* Stacks

start:              * Programmstart
 move #5-1,d0       * hier D0 als Zaehler
schleife:           * Wiederhole
 move d0,-(a7)      *  Register auf Stack
 move #50,d0        *  neu besetzen
 jsr @schreite      *  fuer eine Linie
 move #72,d0        *  und Drehung
 jsr @drehe         *  ausfuehren
 move (a7)+,d0      *  Register von Stack
 dbra d0,schleife   * steht dann in D0
 rts                * bereit.
```

Abb. 3.5.10 Programmbeispiel für die Anwendung eines Stacks

Es gibt nun viele Interruptmöglichkeiten beim 68008. Wir verwenden aber nur drei besonders einfache. Den Interrupt Autolevel 2,5 und 7. Wenn die Leitung NMI auf 0V geht und der Interrupt freigegeben wurde, so holt sich der Prozessor von Adresse $68 ein Langwort. Dieses Langwort ist die Adresse des Interruptprogramms. Bei der Leitung INT wird der Vektor von Adresse $74 geholt, und wenn man beide Leitungen gleichzeitig auf 0 V legt, so wird die Adresse von $7C geladen.

Auf diesen Adressen stehen Werte, die RAM-Adressen bedeuten. Ab diesen RAM-Adressen wiederum ist ein Platz von jeweils 6 Bytes reserviert; dieser Platz ist ausreichend für einen absoluten Sprung. Es ist also möglich, eigene Interruptprogramme zu schreiben, die dann über einen solchen absoluten Sprung aufgerufen werden.

Die anderen Interruptarten sind nicht verwendet. Für die „User Interrupts" müßte eine andere Beschaltung der CPU verwendet werden, und für die anderen Autointerrupts ist der 68000 nötig, denn der 68008 hat die beiden Eingänge IPL0 (Interruptlevel oder Unterbrechungsebene) und 2 schon im IC zusammengeschaltet.

Wenn man auch noch IPL1 damit verbindet, also z.B. die Brücke JMP2 auf der CPU-Baugruppe einlötet (*Abb. 3.5.11*), so wird der Interrupt Autovektor 7 angesprungen. Dieser Vektor hat eine Besonderheit. Bitte die Brücke gleich einlöten, denn sie wird für das folgende Programm gebraucht.

Man unterscheidet zwei Arten von Interrupts. Die maskierbaren Interrupts und die nicht maskierbaren Interrupts. Der Autovektor 7 ist nicht maskierbar. Das heißt, er wird immer ausgelöst. Ferner wird er nur dann ausgelöst, wenn ein Wechsel von 1 auf 0 an den Leitungen stattfindet; er reagiert also nur auf eine Flanke, während die anderen immer reagieren, wenn sie auf 0 V liegen. Die maskierbaren Interrupts reagieren außerdem nur, wenn man die Interrupts freigibt. Für die Regelung, welche Interruptlevels freigegeben sind und welche nicht, gibt es im Statusregister eine sogenannte Unterbrechungsebene. Sie ist normalerweise auf 7 geschaltet, Interrupts mit niedrigeren Nummern sind dann nicht zugelassen. *Abb. 3.5.12* zeigt die Belegung des Statusregisters.

System- oder Userbetrieb

Die Bits 15 bis 8 im Statusregister sind die sogenannten Systembits. Bits 7 bis 0 sind die User-Bits. Im 68008 unterscheidet man die zwei Betriebsarten System- und Userbetrieb. Ist Bit 13 auf 1, so ist der Systembetrieb eingestellt. Dann können alle Befehle des 68008 ausgeführt werden, im User-Mode geht das nicht. Es gibt privilegierte Befehle, die zu einer Exception führen, wenn man sie im User-Mode ausführt. Im Grundprogramm werden auch die Anwender-Programme im System-Mode ausgeführt. Man kann den User-Mode einschalten, indem man das Bit 13 im Statusregister auf 0 setzt. Wenn man den User-Mode einmal eingeschaltet hat, so kann man ihn nicht wieder ausschalten. Der Befehl für das Umschalten gehört zu den privilegierten Befehlen. Nur für die Dauer eines TRAP-Aufrufes oder für Exceptions wird der System-Mode wieder hergestellt.

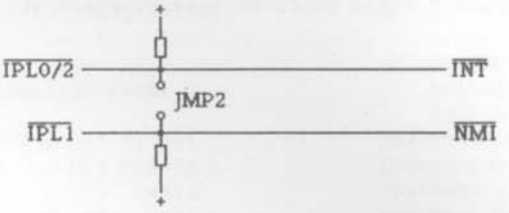
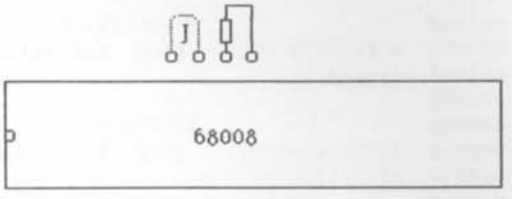


Abb. 3.5.11
Einsetzen der Brücke
in die CPU-Baugruppe



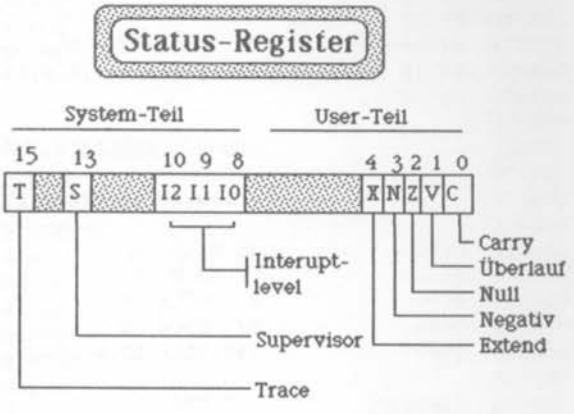


Abb. 3.5.12
Das Status-Register
des 68000/68008

```
Rolf-D.Klein 68000/08 Assembler 4.3 (C) 1984, Seite 1

009C00                  ;*********************************
009C00                  ; MONDLANDEPROGRAMM VERSION 1.0 ;
009C00                  ; VOM 13.12.1983                ;
009C00                  ;*********************************
009C00
009C00                  ; ***** INTERRUPT-BAUSTEIN *****
009C00
00800C                   ORG $800C ; FESTE ADRESSE
00800C                                                    zu Abb. 3.5.13
00800C  4EF9 0000A0A2    JMP INTAUS ; DORTHIN SPRUNG
008012
00A000                   ORG $A000
```

```
Rolf-D.Klein 68000/08 Assembler 4.3 (C) 1984, Seite 2

00A000
00A000                    ; ***** VEREINBARUNGS-BAUSTEIN *****
00A000
= FFFFFF30                TASTEN EQU $FFFFFF30
= 00000001                RECHTS EQU 1    ; BIT MASKEN
= 00000002                LINKS  EQU 2
= 00000004                RAUF   EQU 4
= 00000008                RUNTER EQU 8
00A000
00A000                    ERFOLGTEXT:
00A000  4775742067656C     DC.B 'Gut gelandet',0
00A007  616E64657400
00A00D
00A00D                    BRUCHTEXT:
00A00D  42727563686C61     DC.B 'Bruchlandung',0
00A014  6E64756E6700
00A01A
00A01A                    FAEHRE:
00A01A  08 02 09 00 00     DC.B 8,2,9,0,0,7,3,2,2,4,4,4,4
00A01F  07 03 02 02 04
00A024  04 04 04
00A027  06 06 05 01 00     DC.B 6,6,5,1,0,0,10
00A02C  00 0A
00A02E
00A02E                     DS 0 ; ANGLEICH AUF WORTGRENZE
00A02E
00A02E                    ; VARIABLENVEREINBARUNGEN
00A02E
00A02E  0000              X: DC.W  0
00A030  0000              Y: DC.W  0
00A032  0000              DX: DC.W  0
00A034  0000              DY: DC.W  0
00A036  0000              TREIBST: DC.W  0
00A038
00A038  00000000          TIMERMS: DC.L 0
00A03C  0000              TIMER10TEL: DC.W  0
00A03E                    BUFFER: DS 50
00A0A2
00A0A2                    ; ***** UNTERPROGRAMM-BAUSTEIN *****
00A0A2
00A0A2                    INTAUS:
00A0A2  48E7 1000          MOVEM.L D3,-(A7)
00A0A6  2639 0000A038      MOVE.L TIMERMS,D3
00A0AC  0683 00000014      ADD.L #20,D3
00A0B2  23C3 0000A038      MOVE.L D3,TIMERMS
00A0B8  86FC 0064          DIVU #100,D3 ;  UMRECHNEN IN 10TEL SEC
00A0BC  33C3 0000A03C      MOVE.W  D3,TIMER10TEL
00A0C2  4CDF 0008          MOVEM.L (A7)+,D3
00A0C6  4E73               RTE
00A0C8
00A0C8                                                    zu Abb. 3.5.13
00A0C8                    ZEITAUS:
00A0C8  41F9 0000A03E      LEA BUFFER,A0
```

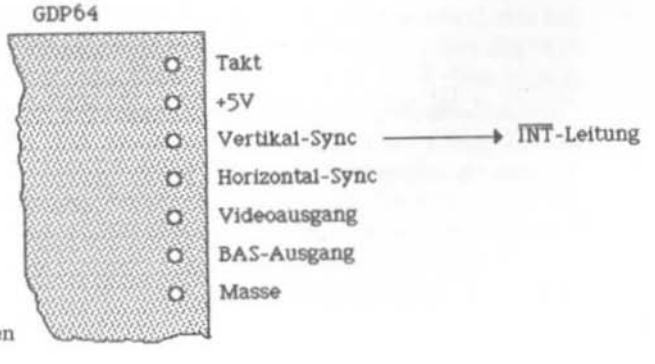
```
Rolf-D.Klein 68000/08 Assembler 4.3 (C) 1984, Seite 3

00A0CE  3039 0000A03C    MOVE.W  TIMER10TEL,D0
00A0D4  4EB9 000015FC    JSR @PRINT4D
00A0DA  41F9 0000A03E    LEA BUFFER,A0
00A0E0  303C 0022        MOVE #$22,D0
00A0E4  323C 0000        MOVE #0,D1
00A0E8  343C 00DC        MOVE #220,D2
00A0EC  4EB9 00001386    JSR @WRITE
00A0F2  4E75             RTS
00A0F4
00A0F4
00A0F4                   MONDFLAECHE:
00A0F4  323C 0000        MOVE #0,D1
00A0F8  343C 0000        MOVE #0,D2
00A0FC  363C 005A        MOVE #90,D3
00A100  4EB9 0000131A    JSR @SET       ;SETZE SCHILDKROETE
00A106  4EB9 00001112    JSR @HIDE      ;ABER NICHT ZEIGEN
00A10C  383C 0031        MOVE #50-1,D4     ;ANZAHL KRATER
00A110                   MLP1:
00A110  4EB9 0000A11C    JSR KRATER
00A116  51CC FFF8        DBRA D4,MLP1
00A11A  4E75             RTS
00A11C
00A11C                   KRATER:
00A11C  303C FFD3        MOVE  #-45,D0
00A120  4EB9 000012F6    JSR @DREHE
00A126  303C 000A        MOVE #10,D0
00A12A  4EB9 00001288    JSR @SCHREITE
00A130  303C FFD3        MOVE #-45,D0
00A134  4EB9 000012F6    JSR @DREHE
00A13A  303C 0006        MOVE #6,D0
00A13E  4EB9 00001288    JSR @SCHREITE
00A144  303C FFD3        MOVE #-45,D0
00A148  4EB9 000012F6    JSR @DREHE
00A14E  303C 000A        MOVE #10,D0
00A152  4EB9 00001288    JSR @SCHREITE
00A158  303C FF1F        MOVE #-45-180,D0
00A15C  4EB9 000012F6    JSR @DREHE
00A162  4E75             RTS
00A164
00A164                   ; ***** PROGRAMM-BAUSTEIN *****
00A164
00A164                   START:
00A164  6100 FF8E        BSR MONDFLAECHE
00A168  33FC 0064        MOVE #100,X
00A16C  0000A02E
00A170  33FC 00C8        MOVE #200,Y                    zu Abb. 3.5.13
00A174  0000A030
00A178  33FC 0003        MOVE #3,DX
00A17C  0000A032
00A180  33FC 0000        MOVE #0,DY
00A184  0000A034
00A188  33FC 012C        MOVE #300,TREIBST
```

```
Rolf-D.Klein 68000/08 Assembler 4.3 (C) 1984, Seite 4

00A18C  0000A036
00A190  42B9 0000A038    CLR.L TIMERMS          ; INITIALISIEREN
00A196  4279 0000A03C    CLR TIMER10TEL
00A19C                  WDH1:
00A19C  6100 FF2A        BSR ZEITAUS
00A1A0  207C 0000A01A    MOVEA.L #FAEHRE,A0
00A1A6  303C 0003        MOVE #$3,D0             ; VERGROESSERUNG=3
00A1AA  3239 0000A02E    MOVE X,D1
00A1B0  3439 0000A030    MOVE Y,D2
00A1B6  4EB9 0000328C    JSR @FIGUR
00A1BC  4A79 0000A036    TST TREIBST
00A1C2  6F00 0052        BLE NOTEST              ; KEIN TREIBSTOFF ?
00A1C6  1039 FFFFFF30    MOVE.B TASTEN,D0
00A1CC  0200 0001        AND.B #RECHTS,D0
00A1D0  6608             BNE.S TST1
00A1D2  5679 0000A032    ADDQ #3,DX
00A1D8  603C             BRA.S NOTEST
00A1DA
00A1DA                  TST1:
00A1DA  1039 FFFFFF30    MOVE.B TASTEN,D0
00A1E0  0200 0002        AND.B #LINKS,D0
00A1E4  6600 000A        BNE TST2
00A1E8  5779 0000A032    SUBQ #3,DX
00A1EE  6026             BRA.S NOTEST
00A1F0
00A1F0                  TST2:
00A1F0  1039 FFFFFF30    MOVE.B TASTEN,D0
00A1F6  0200 0004        AND.B #RAUF,D0
00A1FA  6608             BNE.S TST3
00A1FC  5679 0000A034    ADDQ #3,DY
00A202  6012             BRA.S NOTEST
00A204
00A204                  TST3:
00A204  1039 FFFFFF30    MOVE.B TASTEN,D0
00A20A  0200 0008        AND.B #RUNTER,D0
00A20E  6606             BNE.S NOTEST
00A210  5779 0000A034    SUBQ #3,DY
00A216
00A216                  NOTEST:
00A216  5379 0000A034    SUBQ #1,DY              ;GRAVITATION
00A21C  3639 0000A032    MOVE DX,D3              ; X:=X+DX
00A222  D779 0000A02E    ADD D3,X
00A228  3639 0000A034    MOVE DY,D3              ; Y:=Y+DY
00A22E  D779 0000A030    ADD D3,Y
00A234  5379 0000A036    SUBQ #1,TREIBST         ; TREIBSTOFF VERINGERN
00A23A
00A23A  323C 0004        MOVE #5-1,D1           ; WARTESCHLEIFE
00A23E                  WARTE20MS:
00A23E  3E3C 001C        MOVE #!SYNC,D7
00A242  4E41             TRAP #1
00A244  6700 FFF8        BEQ WARTE20MS
00A248  51C9 FFF4        DBRA D1,WARTE20MS
00A24C
```

zu Abb. 3.5.13

```
00A24C  0C79 0005          CMP #5,Y                ; HOEHE DER OBERFLAECHE UEBER 0
00A250  0000A030
00A254  6E00 FF46          BGT WDH1                ; BERUECKSICHTIGEN
00A258                                             ;
00A258  303C 0003          MOVE #$3,D0             ; FAEHRE AM SCHLUSS NOCHMALS ZEIGEN
00A25C  3239 0000A02E      MOVE X,D1
00A262  3439 0000A030      MOVE Y,D2
00A268  4EB9 0000328C      JSR @FIGUR              ; FIGUR AUFRUFEN
00A26E  3039 0000A032      MOVE DX,D0
00A274  6A02               BPL.S FF1
00A276  4440               NEG D0
00A278                     FF1:                    ; !DX! < 2
00A278  0C40 0002          CMP #2,D0
00A27C  6C18               BGE.S BRUCH
00A27E  3039 0000A034      MOVE DY,D0
00A284  6A02               BPL.S FF2
00A286  4440               NEG D0
00A288                     FF2:                    ; !DY! < 4
00A288  0C40 0004          CMP #4,D0
00A28C  6C08               BGE.S BRUCH
00A28E  41F9 0000A000      LEA ERFOLGTEXT,A0
00A294  6006               BRA.S ENDE
00A296
00A296                     BRUCH:
00A296  41F9 0000A00D      LEA BRUCHTEXT,A0
00A29C                     ENDE:
00A29C  303C 0033          MOVE #$33,D0            ; SCHRIFTGROESSE
00A2A0  323C 0000          MOVE #0,D1              ; X-POSITION
00A2A4  343C 0082          MOVE #130,D2            ; Y-POSITION
00A2A8  4EB9 000013B6      JSR @WRITE
00A2AE  6100 FE18          BSR ZEITAUS             ; LETZTER STAND
00A2B2  4EB9 00000A00      JSR @CI                 ; WARTEN AUF TASTE
00A2B8  4EB9 00000DA0      JSR @CLR                ; LOESCHEN
00A2BE  6000 FEA4          BRA START               ; UNBEGRENZT
00A2C2
00A2C2                     ; ***********
00A2C2
00A2C2                     END

008F94  Ende-Symboltabelle
```

Oben: Abb. 3.5.13
Das Mondlandeprogramm mit Zeiteinblendung. Achtung! Die $\overline{\text{Int}}$-Leitung erst nach der Übersetzung aktivieren; siehe Text



Hier wird die $\overline{\text{Int}}$-Leitung angeschlossen

Das Umschalten der Unterbrechungsebene

Möchte man versuchsweise die jetzt nicht genehmigten Interruptmöglichkeiten austesten, so muß man den Interruptlevel entsprechend ändern. Eine Veränderung der Bits ist mit MOVE, EOR, AND und OR-Befehlen möglich, wenn man als Ziel SR angibt. MOVE #$2700,SR setzt den Interruptlevel auf 7 zurück. Achtung, wenn man das Statusregister im Einzelschritt sieht, so ist auch Bit 15 gesetzt, denn dann wird das Trace-Bit gesetzt, um Befehle im Einzelschritt durchlaufen zu können. Wenn man das Tracebit setzt, so wird ein Befehl ausgeführt und anschließend eine Exception auf den Trace-Vektor durchgeführt.

Im privilegierten Mode, also im User-Mode, kann man nur noch die Bedingungsflags setzen, also z. B. mit MOVE ...,CCR; der schreibende Zugriff auf das Statusregister ist verboten.

Der Einbau der Interrupttechnik in das Mondlandeprogramm

Abb. 3.5.13 zeigt das vollständige Programm „Mondlandung mit Zeitnahme".

Es beginnt mit dem Interrupt-Baustein. Mit dem ORG $800C - Befehl wird erreicht, daß der nachfolgende Befehl, ein Sprung, an die Stelle im RAM gelegt wird, auf die der Sprung beim Interrupt Level 7 automatisch erfolgt. Dort kann man aber nur diesen Sprung hinlegen, das eigentliche Interruptprogramm muß woanders stehen, es hat hier keinen Platz. Darum folgt eine weitere ORG-Anweisung. Dann folgt der Vereinbarungs-Baustein, wie schon im vorherigen Abschnitt beschrieben. Neu ist, neben ein paar Variablen, erst der Unterprogramm-Baustein. Dort befindet sich die Routine INTAUS; sie wird alle 20 ms durch einen Interrupt angesprungen. Zunächst wird das Register D3 gerettet, da es im Programmteil verwendet wird. Dann wird der alte Zeitwert der Variable TIMERMS geladen, 20 addiert und dann wieder zurückgespeichert. Für die Ausgabe wird der Wert dann noch durch 100 dividiert, denn eine Ausgabe in 10tel Sekunden genügt uns. Dieser Wert wird in TIMER10TEL gespeichert. Bevor nun der Rücksprung per RTE ins Hauptprogramm erfolgen kann, ist noch der alte Wert des Registers D3 vom Stack zurückzuladen.

Wie bereits besprochen, bedeutet RTE „return from exception", also Rückkehr von der Ausnahme, und unterscheidet sich vom RTS bei Unterprogrammen dadurch, daß es das alte Statusregister zurückholt, dessen Inhalt von der CPU automatisch beim Interrupt gerettet wurde. Man darf also hier nicht RTS verwenden, sonst stürzt das Programm ab.

Die anderen Programmteile sind bekannt. Die Zeitausgabe erfolgt im Hauptprogramm einmal gleich zu Beginn des Spielblocks mit ZEITAUS und dann am Schluß des Programms, vor dem Aufruf von CI.

Die Inbetriebnahme

1. Gemäß *Abb. 3.5.14* wird eine Verbindung vom VSYNC-Ausgang der GDP zum INT-Quer des Busses hergestellt. Die Leitung darf aber nicht direkt an die Stiftleiste gelötet werden, sondern muß über einen Schalter geführt werden. Damit kann man verhindern, daß der Interrupt sofort nach dem Einschalten des Computers ausgelöst wird. Wäre dies der Fall, so würde das VSYNC-Signal einen Interrupt auslösen und somit die Abarbeitung einer Interruptroutine starten, die es noch gar nicht gibt.
2. Die Brücke an der CPU muß auch eingesetzt sein, wie es Abb. 3.5.11 zeigt, sonst wird nicht der nicht maskierbare Interrupt ausgelöst und das Programm funktioniert nicht.
3. Nun kann man den Computer einschalten und das Programm eingeben. Diesmal wieder ab Adresse $9000 als Textstart im Editor. Wenn man den Schalter, der die INT-Leitung am Bus mit VSYNC auf der GDP-Karte verbindet, geschlossen hat, so meldet sich das Grundprogramm nicht, also bitte den Schalter vorher öffnen.
4. Programm mit dem Assembler wie gewohnt übersetzen.
5. Wenn man das Programm startet, so wird die Mondlandung ohne Zeitnahme wie bisher funktionieren.
6. Interrupt einschalten (Verbindung VSYNC --- INT herstellen).
7. Wenn man die Leertaste drückt, wird das Programm neu gestartet, jetzt muß eine Zeitnahme erfolgen.

Ruft man nun das Grundprogramm wieder auf, z. B. durch RESET, so bleibt der Interrupt aktiv. Man kann z. B. durch „ANSEHEN" oder mit „IO LESEN" die Speicherzelle „TIMERMS" ansehen und wird feststellen, daß sie ihren Wert dauernd ändert.

Aufgaben:

1. Was passiert, wenn man den Interrupt Autolevel 5 verwendet? (Brücke JMP 2 offenlassen und SR mit einem kleineren Level belegen ($2000 z. B.), ORG $8006 verwenden). Warum stimmt die Zeitnahme nicht mehr? Was muß man schaltungstechisch bei der Interruptauslösung ändern, oder was im Interruptprogramm?
2. Denken Sie sich weitere Einsatzmöglichkeiten für Interrupts aus.

Einblenden der Uhrzeit ins Grundprogramm

Abschließend soll noch ein Beispiel für die Leistungsfähigkeit des Interrupts gebracht werden. Eine Uhr soll im Grundprogramm eingeblendet werden. Auch hier braucht man wirklich den Interrupt, denn das Grundprogramm läßt sich nicht so leicht umkonstruieren.

Abb. 3.5.15 zeigt das fertige Programm. Um das Programm verwenden zu können, muß man bei der Übersetzung genauso vorgehen, wie bei der Mondlandung. Die Verbindung INT mit VSYNC darf erst dann hergestellt werden, wenn das Programm fertig übersetzt ist. Das Programm selbst wird nicht mit „2=starten" gestartet, sondern es wirkt, sobald man die Leitung verbindet. Achtung, die Brücke (JMP2) auf der CPU-Baugruppe muß natürlich auch vorhanden sein!

```
Rolf-D.Klein 68000/08 Assembler 4.3 (C) 1984, Seite 1


009C00
009C00                  ***************************
009C00                  * UHRZEIT EINBLENDEN      *
009C00                  ***************************
009C00
009C00                  * INTERRUPT - BAUSTEIN
009C00
00800C                   ORG $800C        * FESTE ADRESSE
00800C  4EF9 00009C70    JMP INTAUS       * DORT INT-PROGRAMM
008012
008012
009C00                   ORG $9C00        * Z.B. HIER PROGRAMM
009C00
009C00                  * VEREINBARUNGS-BAUSTEIN
009C00
= FFFFFF70              GDP EQU $FFFFFF70
009C00
009C00                  GDPREGISTER: DS.B 15  * HIER ALLE REGISTERWERTE MERKEN
009C0F                  BUFFER:   DS.B 80     * AUSGABE BUFFER
009C5F  00                        DS 0        * BYTE ANGLEICH
009C60                                        *
009C60  0000            MILLISEK: DC.W 0      * UHRZEIT DURCH KONSTANTEN
009C62  0000            SEKUNDEN: DC.W 0      * VOREINSTELLBAR
009C64  0008            MINUTEN:  DC.W 08     *
009C66  0016            STUNDEN:  DC.W 22     *
009C68
009C68                  * UNTERPROGRAMM-BAUSTEIN
009C68
009C68                  TRENNUNG:         * ZEICHEN ZWISCHEN DER ZEITAUSGABE
009C68  10FC 003A        MOVE.B #':',(A0)+
009C6C  4210             CLR.B (A0)
009C6E  4E75             RTS
009C70
009C70                  * PROGRAMM-BAUSTEIN
009C70
009C70                  INTAUS:           * IST DIESMAL AUCH DAS
009C70                                    * HAUPTPROGRAMM
009C70  48E7 FFFE        MOVEM.L D0-D7/A0-A6,-(A7)    * ALLES RETTEN
009C74  4EB9 00000D5C    JSR @WAIT        * WARTEN BIS GDP FERTIG
009C7A  43F9 00009C00    LEA GDPREGISTER,A1   * ZIEL
009C80  41F9 FFFFFF71    LEA GDP+1,A0     * DANN ALLE GDP-REGISTER RETTEN
009C86  363C 000E        MOVE #15-1,D3    * NUR COMMANDOPORT NICHT
009C8A                  RETTE6:
009C8A  12D8             MOVE.B (A0)+,(A1)+  * OK TRANSPORT
009C8C  51CB FFFC        DBRA D3,RETTE6
009C90  0679 0014        ADD #20,MILLISEK    * ERST MS BILDEN
009C94  00009C60
009C98  0C79 03E8        CMP #1000,MILLISEK  * WENN <1000 DANN WEITER
009C9C  00009C60
009CA0  6D00 00A6        BLT INTO            * SONST SEKUNDEN BEARBEITEN
009CA4  4279 00009C60    CLR MILLISEK        * BEREICH 0..999
009CAA  0679 0001        ADD #1,SEKUNDEN     * SEKUNDE NEU
```

zu Abb. 3.5.15

```
olf-D.Klein 68000/08 Assembler 4.3 (C) 1984, Seite 2

09CAE  00009C62
09CB2  0C79 003C        CMP #60,SEKUNDEN     * 0..59 ERLAUBT
09CB6  00009C62
09CBA  6D00 0034        BLT INT1             * JA, DANN WEITER
09CBE  4279 00009C62    CLR SEKUNDEN         * SONST MINUTEN
09CC4  0679 0001        ADD #1,MINUTEN       * ERHOEHEN
09CC8  00009C64
09CCC  0C79 003C        CMP #60,MINUTEN      * AUCH 0..59
09CD0  00009C64
09CD4  6D00 001A        BLT INT1             * WEITER WENN IM BEREICH
09CD8  4279 00009C64    CLR MINUTEN          * SONST UEBERTRAG AUF STUNDEN
09CDE  0679 0001        ADD #1,STUNDEN       * 0..23 ERLAUBT
09CE2  00009C66
09CE6  6D00 0008        BLT INT1             * JA, DANN WEITER
09CEA  4279 00009C66    CLR STUNDEN          * EINE WEITERER UEBERTRAG
09CF0                  INT1:                 * NUN AUSGABE AUF DEN BILDSCHIRM DURCHFUEHREN
09CF0                                        * ABER NUR ALLE SEK. DURCHFUEHREN
09CF0  41F9 00009C0F    LEA BUFFER,A0        * ZIEL DER AUSGABE
09CF6  3039 00009C66    MOVE STUNDEN,D0      * ALLES IN ASCII UMWANDELN UND ABLEGEN
09CFC  4EB9 000015FC    JSR @PRINT4D         * 0..23
09D02  6100 FF64        BSR TRENNUNG
09D06  3039 00009C64    MOVE MINUTEN,D0
09D0C  4EB9 000015FC    JSR @PRINT4D
09D12  6100 FF54        BSR TRENNUNG
09D16  3039 00009C62    MOVE SEKUNDEN,D0
09D1C  4EB9 000015FC    JSR @PRINT4D        * 21:47:0 AUSGEBEN, Z.B.
09D22  363C 0009        MOVE #10-1,D3       * REST AUFFUELLEN
09D26                  INTLP1:
09D26  10FC 0020        MOVE.B #' ',(A0)+   * MIT LEERZEICHEN
09D2A  51CB FFFA        DBRA D3,INTLP1      * DA UNTERSCHIEDLICH VIELE ZEICHEN
09D2E  4210             CLR.B (A0)          * ENDE MARKIEREN
09D30  41F9 00009C0F    LEA BUFFER,A0       * UND DANN AUSGEBEN
09D36  103C 0022        MOVE.B #$22,D0      * SCHRIFTGROESSE 2,2
09D3A  323C 0190        MOVE #400,D1        * X-KOORDINATE
09D3E  343C 0001        MOVE #1,D2          * Y-KOORDINATE
09D42  4EB9 00001386    JSR @WRITE          * UND AUSGABE DURCHFUEHREN
09D48                  INT0:                * ALLE 20 MS, ERFOLGT HIER EIN EINSPRUNG
09D48                  ********** INT--PROGRAMM ENDE, JETZT REGISTER ZURUECK **********
09D48  4EB9 00000D5C    JSR @WAIT       * DANN WIEDER ZURUECK DAMIT
09D4E  43F9 00009C00    LEA GDPREGISTER,A1   * QUELLE
09D54  41F9 FFFFFF71    LEA GDP+1,A0    * DANN ALLE GDP-REGISTER LADEN
09D5A  363C 000E        MOVE #15-1,D3   * NUR COMMANDOPORT NICHT
09D5E                  LADEG:
09D5E  10D9             MOVE.B (A1)+,(A0)+  * OK TRANSPORT
09D60  51CB FFFC        DBRA D3,LADEG       * DURCHFUEHREN
09D64  4CDF 7FFF        MOVEM.L (A7)+,D0-D7/A0-A6    * UND ALTE INHALT WIEDER ZURUECK
09D68  4E73             RTE
09D6A
09D6A
09D6A
09D6A
09D6A
09D6A
09D6A
09D6A
09D6A
09D6A
09D6A

08B80  Ende-Symboltabelle
```

Abb. 3.5.15 Programm zur Einblendung einer Uhrzeit in das Grundprogramm

Es erscheint dann z. B. die Uhrzeit in die Menues nach *Abb. 3.5.16* eingeblendet. Alle Menues sind betroffen. Das Interrupt-Programm bietet noch eine Besonderheit. Bei „INTAUS" werden zunächst alle Register des 68000/8 gerettet. Denn wir verwenden im Interrupt-Programm Unterprogramme aus dem Grundprogramm und wissen somit nicht, welche Register verändert werden. Dann folgt ein Aufruf „JSR @WAIT"; damit wird gewartet, bis der Graphik-Prozessor seinen letzten Befehl ausgeführt hat. Dann werden alle Register des Graphik-Prozessors gerettet. Wir haben ja noch eine CPU im Computer, den Graphik-Prozessor. Da wir eine Uhrzeit einblenden, werden wir ihn zur Ausgabe benötigen. Die Register des Graphik-Prozessors können aber im Grundprogramm gerade mit wichtigen Werten belegt worden sein, und bei einer Unterbrechung muß sichergestellt sein, daß sie nachher genauso aussehen, wie vor der Unterbrechung.

Im Programm selbst kann man dann die Ausgabe beliebig gestalten, und vor der Rückkehr werden die Register wieder zurückgeladen. Es gibt beim Interruptprogramm noch einiges zu beachten, so z.B. müssen natürlich die verwendeten Unterprogramme unterbrechungsfähig sein. Verwendet z.B. ein Unterprogramm neben Registern auch Speicherzellen als Variable, und wird dieses Unterprogramm dann auch vom Interruptprogramm aufgerufen, dann werden dabei auch die Speicherzellen verändert. Trat nun der Interrupt während der Abarbeitung dieses Unterprogramms auf, also dann, wenn es gerade vom Hauptprogramm aufgerufen wurde, so gehen dabei die – möglicherweise noch wichtigen – Daten in den Variablen verloren. Die entsprechende Speicherzelle wäre also ebenso zu retten, wie wir das bei den Registern getan haben. Man sollte daher Interrupts nur dann verwenden, wenn man die Sachlage genau durchdenken kann, man also alle verwendeten Unterprogramme kennt.

Aufgaben:

1. Erstellen Sie das Struktogramm zu dem gegebenen Uhrenprogramm.
2. Die Uhreinblendung funktioniert nicht immer störungsfrei, manchmal blinkt sie auch (z.B. im Editor). Versuchen Sie, diesen Fehler zu beheben. Die Behebung ist nicht ganz einfach. Dazu muß man neue Befehle verwenden, die man z. B. aus einem späteren Kapitel heraussuchen kann. Kontrollieren Sie vor der Verwendung der Unterprogramme auch, ob diese Speicherzellen verändern, und retten Sie deren Inhalte gegebenenfalls.

   Alle Befehle, mit denen man die Bildseiten-Umschaltung beeinflussen kann, sollten Sie besonders betrachten.
3. Schreiben Sie das Standuhr-Programm auf Interrupt-Betrieb um, dann ist eine quarzgenaue Uhrzeit gegeben. Schreiben Sie auch ein Hilfsprogramm zum einfachen Stellen der Uhr.

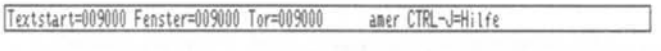
3.6 Der Zufallsgenerator

In der Natur gibt es viele Prozesse, die zufallsbedingt ablaufen. Um solche Prozesse zu simulieren, benötigt man einen sogenannten Zufallsgenerator. Er hat die Aufgabe,

Abb. 3.5.16
Die Uhrzeit im Grundprogramm-Menue

```
 org $9c00

start:
 move #4,d0       * Anzahl der Schritte
 jsr @schreite    * und vorwaerts
 move #360,d0     * Limit angeben
 jsr @rnd         * Zufalls-Zahl 0..359
 jsr @drehe       * zufaellige Drehung
 jsr @csts        * bis Taste gedrueckt
 beq start        * wiederholen
 rts              * Ende
```

Abb. 3.6.1
Programm:
Zufällige Bewegung einer Schildkröte



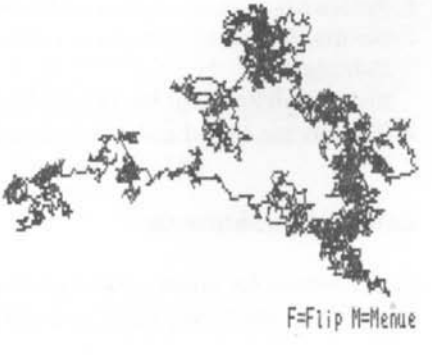


Abb. 3.6.2
Ein möglicher Bildschirminhalt

zufällige Zahlen zu liefern, mit denen man dann arbeiten kann. So könnte man zum Beispiel ein Würfelspiel simulieren, oder den Lauf einer Roulettkugel.

Ein Zufallsgenerator befindet sich, fertig eingebaut, im Grundprogramm der Version 4.3. Man kann ihn durch den Unterprogrammaufruf „JSR @RND" erreichen. Im Register D0.W steht dazu der obere Grenzwert, also die Zahl, die nicht mehr als Zufallszahl geliefert werden soll. Im Register D0 erscheint nach dem Aufruf eine zufällige Zahl zwischen 0 und dem angegebenen Wert -1.

Die wandernde Schildkröte

Die Schildkrötengraphik eignet sich hervorrangend um den Zufallsgenerator auszuprobieren. Unsere Schildkröte soll einen zufälligen Weg laufen. Dazu schreitet sie immer ein Stück und dreht sich dann um einen zufälligen Wert. Die Lösung dieser Aufgabe zeigt *Abb. 3.6.1*. Die Schildkröte soll um 4 Schritte vorwärts gehen und sich anschließend um einen Winkel zwischen 0 und 359° drehen. Dazu wird das Register D0.W mit dem Wert 360 geladen. Der Zufallsgenerator liefert dann einen Wert zwischen 0 und 359. Dieser Wert wird dann als Parameter direkt dem „DREHE"-Befehl zugeführt. Ein mögliches Ergebnis zeigt *Abb. 3.6.2*, und wenn man das Programm eine Weile laufen läßt, ergibt sich z. B. *Abb. 3.6.3*. Das Bildfeld wird aber wohl meist nicht ganz gefüllt, da die Schildkröte auch außerhalb des sichtbaren Bereichs laufen kann. Wenn man die Schrittweite vergrößert, so entstehen gröbere Figuren. Die *Abb. 3.6.4* zeigt unser Programm in dieser abgewandelten Form; zusätzlich wird noch der Winkel in gröberen Schritten verändert. Der Zufallsgenerator liefert hier einen Wert zwischen 0 und 7. Durch die nachfolgende Multiplikation mit dem Wert 45 wird erreicht, daß sich die Schildkröte nur um Winkel wie 0,45,90,135,180,225,270 oder 315° drehen kann.

Das gezeichnete Bild kann dann wie die *Abb. 3.6.5* aussehen. Diesmal zeichnet die Schildkröte geometrische Muster.

Aufgaben:

1. Lassen Sie die Schildkröte nur um 0,90,180 oder 270 Grad drehen. Was ergibt sich als Muster?
2. Prüfen Sie, was passiert, wenn man im ersten Programm (Abb 3.6.1) als Bereich 0..1 für die Zufallszahl vorgibt. Erklären Sie das Zustandekommen dieser kreisförmigen Gebilde.
   Bis zu welcher Zahl für D0 ergeben sich kreisförmige Gebilde?
3. Wählen Sie auch für die Schrittweite einen zufälligen Wert.

Zufällige Punktmuster

Entscheidend für einen guten Zufallsgenerator ist eine gute Verteilung der Zufallszahlen. Einen harten Test kann man mit dem Programm nach *Abb. 3.6.6* durchführen. Es

Abb. 3.6.3
Der Bildschirm nach einiger Zeit

```
 org $9c00

start:
 move #10,d0     * Anzahl der Schritte
 jsr @schreite   * und vorwaerts
 move #8,d0      * Limit angeben
 jsr @rnd        * Zufalls-Zahl 0..7
 muls #45,d0 * 0,45,90,135,180,225,270,315
 jsr @drehe      * zufaellige Drehung
 jsr @csts       * bis Taste gedrueckt
 beq start       * wiederholen
 rts             * Ende
```

Abb. 3.6.4
Programm mit 8 verschiedenen Winkeln

```
Textstart=009000 Fenster=009000 Tor=009000      amer CTRL-J=Hilfe
```

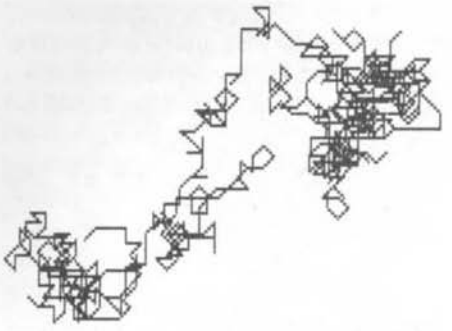


Abb. 3.6.5
Ein mögliches Programmergebnis

F=Flip M=Menue

```
▒
▒
▒

start:
 move #512,d0      * x-Koordinate max
 jsr @rnd          * bestimmen
 move d0,d1        * und merken
 move #256,d0      * y-Koordinate
 jsr @rnd          * bestimmen
 move d0,d2        * fuer MOVETO
 jsr @moveto       * positionieren
 move.b #$80,d0    * Befehl fuer Einzelpunkt
 jsr @cmd          * an GDP ausgeben
 jsr @csts         * bis Taste gedrueckt
 beq start         * wiederholen
 rts

▒
▒
▒
▒
▒
```

Abb. 3.6.6
Test für Zufallszahlen

Textstart=009000 Fenster=009000 Tor=009000 amer CTRL-J=Hilfe

wird zunächst eine Zufallszahl zwischen 0 und 511 ausgewählt, dann eine Zufallszahl zwischen 0 und 255. Die erste Zufallszahl dient als x-Koordinate und die zweite als y-Koordinate. Mit „MOVETO" wird die Position auf dem Bildschirm angewählt, und mit „CMD" wird ein Befehl an den GDP ausgegeben. Der Wert $80 ist dabei der Befehl für „Punkt setzen". An die ausgewählte Position wird also ein Punkt gesetzt.

Abb. 3.6.7 zeigt ein mögliches Ergebnis. Die Punkte sind gleichmäßig über den Bildschirm verteilt.

Der Bildschirm wird sich im Laufe der Zeit immer mehr füllen, bis er schließlich fast ganz hell ist.

Der Zufallsgenerator im Grundprogramm liefert natürlich keine echten Zufallszahlen, sondern sogenannte Pseudo-Zufallszahlen. Diesen Pseudo-Zufallszahlen sieht man es auf ersten Blick nicht an, daß eine Gesetzmäßigkeit vorliegt, das ist ja auch Sinn der Sache. Die Erzeugung geschieht mit Hilfe von Schiebebefehlen und Exklusiv-Oder-Verknüpfungen, sowie mit Hilfe der Funktion SIN, die dafür ausgezeichnet geeignet ist. Dabei ist die Erzeugung von gut verteilten Pseudo-Zufallszahlen nicht ganz einfach. Es gibt dazu sehr viel Literatur, und wer sich dafür interessiert, der kann sich im Literaturverzeichnis des Anhangs darüber informieren.

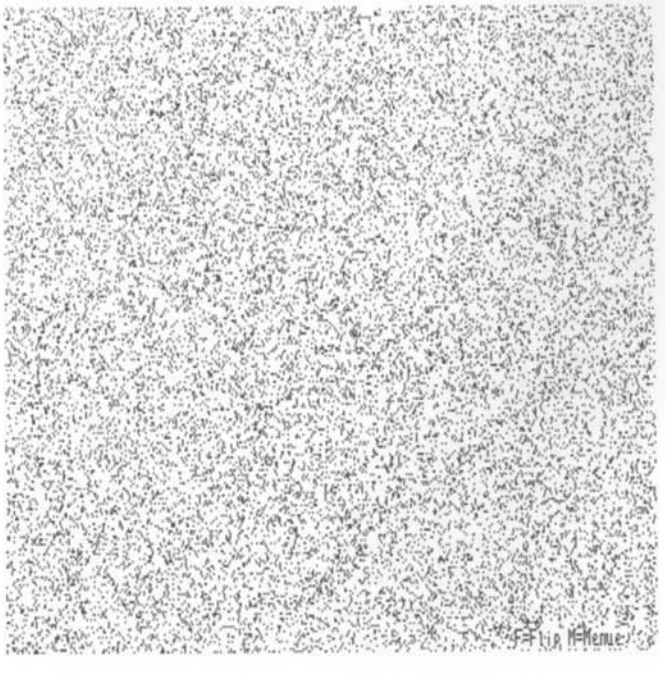


Abb. 3.6.7
Eine gute Verteilung der Punkte

4 Pascal/S

In diesem Kapitel wird die Sprache Pascal auf dem NDR-KLEIN-Computer angewendet. Bei der verwendeten Version Pascal/S handelt es sich nicht um den vollen Sprachschatz der Sprache Pascal, sondern um eine Teilmenge. Dabei sind insbesondere Funktionen wie die Dateiverwaltung nicht vorhanden.

4.1 Die Sprache Pascal

Anfang der 70er Jahre wurde von Prof. Niklaus Wirth an der ETH Zürich die Programmiersprache Pascal aus der Taufe gehoben. Welche Vorteile höhere Programmiersprachen wie Pascal gegenüber der Programmierung in Assembler haben, werden wir in den nächsten Abschnitten dieses Kapitels erkennen.

Bisher haben wir in Assembler programmiert. Damit kann man sehr schnelle Programme schreiben, doch hat das auch einen Haken. Große Programme können leicht unübersichtlich werden, und man kann die Programme nicht auf Rechner mit anderen Prozessoren oder einer anderen Adreßbelegung übertragen. Eine höhere Programmiersprache ist unabhängig vom Prozessor und arbeitet damit auf unterschiedlichen Rechnern.

Dann gibt es in höheren Programmiersprachen direkt Anweisungen wie IF, THEN, ELSE oder REPEAT, UNTIL. Dies sind Anweisungen, die es uns erlauben, die bisher kennengelernten Struktogrammformen unmittelbar in ein Programm umzusetzen. Die aufwendige Formulierung im Assemblercode entfällt.

Eine andere Aufgabe für eine höhere Programmiersprache ist die Bereitstellung von Datenstrukturen. Darunter sind z. B. ein- oder mehrdimensionale Felder zu verstehen, die in Assembler nur mit Mühe realisiert werden können. Moderne Programmiersprachen wie Pascal können auch mathematischen Beweisführungen standhalten. Wie wir bereits kennengelernt haben, ist es eine schwierige Aufgabe, ein mathematisches Problem nach der Umsetzung in ein Assemblerprogramm noch nachzuvollziehen. Hier ist es eine wesentliche Erleichterung, in einer strukturierten Programmiersprache zu arbeiten.

Wie unterscheiden sich nun diese höheren Programmiersprachen in ihrer Arbeitsweise von einem Assembler? Wie wir bereits wissen, kann ein Prozessor nur seine eigenen Maschinenbefehle verstehen. Der Assembler hatte mit den von uns eingegebenen Mnemonics unmittelbar einen solchen Maschinencode erzeugt. Jedem eingegebenen Kürzel, wie z. B. MOVE D1,D0, war dabei ein bestimmter Wert zugewiesen. Für die Arbeit mit höheren Programmiersprachen, wie es Pascal ist, gibt es nun zwei grundlegende Arbeitsweisen, den Compiler und den Interpreter. Beide Einrichtungen sind Hilfsprogramme, die es ermöglichen, den eingegebenen Quellcode durch den Prozessor abzuarbeiten.

Ein Compiler, der eine höhere Programmiersprache übersetzen soll, hat etwas mehr zu tun als ein Assembler, der nur für jedes Kürzel aus einer Tabelle einen bestimmten Code suchen muß. Sehen wir uns zum Erkennen der Arbeitsweise das Beispiel der Variablenzuweisung A1:=B+5 an. Einer Variablen A1 soll die Summe der Variablen B und des Wertes 5 zugewiesen werden. Der Compiler muß diese Anweisung zunächst in Assemblercode übersetzen. Daraus ergeben sich die Anweisungen, die in *Abb. 4.1.1* gezeigt sind. Danach muß der Assembler gestartet werden, der diese Anweisungen in den Maschinencode übersetzt. Es gibt auch Compiler, die den Assembler gleich beinhalten und nicht den Umweg über einen lesbaren Assemblertext gehen, sondern gleich Maschinencode erzeugen. Soviel zunächst zum Compiler; er übersetzt also einen Text mit oder ohne Hilfe des Assemblers in ein lauffähiges Programm.

Der Interpreter dagegen liest eine Anweisung und führt dann gleich den Befehl aus. Es wird also z.B. die Addition durchgeführt und das Ergebnis abgespeichert. Dann holt sich der Interpreter den nächsten Befehl usw. Interpreter werden z.B. bei der Sprache Basic eingesetzt, sind aber für Pascal nicht brauchbar.

Der Compiler übersetzt ein Programm zunächst vollständig, bevor es gestartet wird, der Interpreter übersetzt immer nur ein kleines Stück, führt es dann direkt aus, ohne Maschinencode dafür zu erzeugen, übersetzt dann das nächste Stückchen und so weiter. Das hat den Nachteil, daß die Ausführung eines Programms länger dauert als beim Compiler, da ja ständig übersetzt werden muß. Soll z. B. eine Schleife 100 mal durchlaufen und dabei je eine Addition ausgeführt werden, so muß die Addition 100 mal übersetzt werden.

Es gibt noch eine Mischung aus beiden Verfahren. Den sogenannten Zwischencode-Compiler, bei Pascal spricht man vom PCODE-Compiler. Dabei wird das Programm wie beim Compiler zunächst vollständig übersetzt. Es wird aber nicht der Code für den Mikroprozessor erzeugt, sondern ein Code, der für die Sprache besonders geeignet ist. Wenn dieser Code dann erzeugt ist, wie *Abb. 4.1.2* zeigt, wird er von einem Interpreter ausgeführt, der sich nun PCODE für PCODE holt und ausführt. Diese Mischform ist wesentlich schneller als der reine Interpreter, aber langsamer als der Compiler, der direkt Maschinencode für den Prozessor erzeugt (auch NATIV-Code-Compiler genannt). Der PCODE-Compiler hat den Vorteil, daß man ihn leicht für andere Computer verwenden kann, da man nur den Interpreter neu schreiben muß, die Erzeugung des PCODE aber gleich bleibt. Für die Sprache Pascal gibt es zahlreiche PCODE-Compiler, sicher mit ein Grund für die starke Verbreitung von Pascal.

Auch wir haben einen solchen PCODE-Compiler für den NDR-KLEIN-Computer in EPROMs zur Verfügung. Zum Betrieb benötigt man zusätzlich einen 32 KBytes RAM-Speicher, der zusammen mit dem EPROM-Satz auf einer ROA64-Baugruppe Platz findet. Die *Abb. 4.1.3* zeigt die Anordnung auf der ROA-Baugruppe. Die Karte wird über die Adresse $10000 angesprochen, es sind darum die Brücken 17, 18 und 19 eingesetzt, die Brücke 16 bleibt frei.

Wer schon einen großen RAM-Speicher in diesem Adreßbereich hat (z. B. von $8000 bis $1FFFF), kann sich den Kauf der RAM-Bausteine natürlich sparen und adressiert die Baugruppe einfach auf einer freien Adresse, z. B. $90000. Der Pascal-Compiler ist relokativ, daß heißt, er kann in jedem Adreßbereich laufen. Nur für den RAM-Bereich wir der Bereich von Adresse $18000 bis $1FFFF vorausgesetzt.

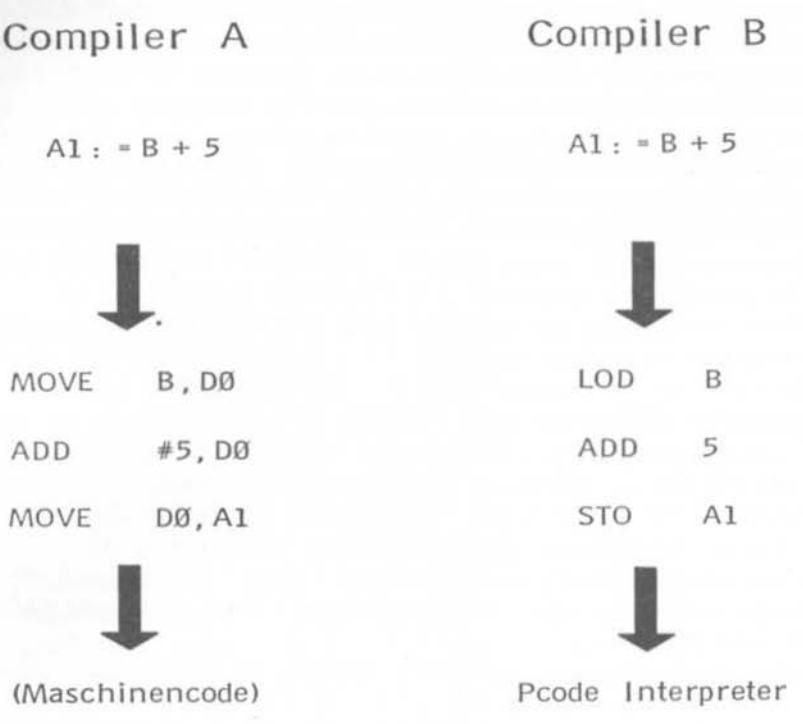


Abb. 4.1.1 Compiler Typ A

Abb. 4.1.2 Compiler Typ B

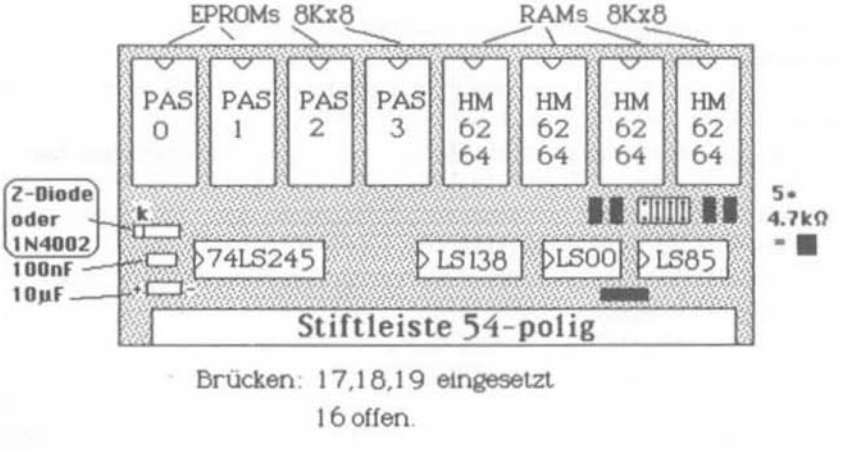


Abb. 4.1.3 Die Anordnung der Pascal-EPROMs auf der ROA64-Baugruppe

```
Name   : PASCAL/S
Start  : 0900F8
Laenge : 008000
```

Abb. 4.1.4
Bei Aufruf der Bibliothek
meldet sich der Pascal-Compiler

```
Starten J = JA
cr = weiter, M = Menue
```

```
program kleinertest(input,output);
 var a:real;
begin
 readln(a);
 writeln(a,' ',sin(a),' ',sqrt(a));
end.
```

Textstart=009000 Fenster=009000 Tor=009000 einf amer CTRL-J=Hilfe

Abb. 4.1.5 Ein kleiner Test

```
##############################
# PASCAL/S  Pcode-Compiler V3.1 #
# (C) 1984 Rolf-Dieter Klein    #
# Version nach PASCAL/S von     #
# N.Wirth 1976, E.T.H Zuerich   #
##############################

    0
    0  program kleinertest(input,output);
    0   var a:real;
    0  begin
    0   readln(a);
    3   writeln(a,' ',sin(a),' ',sqrt(a));
   16  end.

Start PCODE-Interpreter 3.1
3.1
   3.100000000000e+00     4.158062545373e-02     1.760681686165e+00
```

Abb. 4.1.6 Bildschirm nach Eingabe des Wertes 3.1 F=Flip M=Menue

```
##############################
# PASCAL/S  Pcode-Compiler V3.1 #
# (C) 1984 Rolf-Dieter Klein   #
# Version nach PASCAL/S von    #
# N.Wirth 1976, E.T.H Zuerich  #
##############################

   0
   0  program kleinertest(input,output);
   0   var a:real
   0  begin
####       ↑15
   0   readln(a);
   3   writeln(a,' ',sin(a),' ',sqrt(a));
  16  end.

 key words
15 ;
▌
                                          F=Flip M=Menue
```

Abb. 4.1.7
Fehlermeldungen von Pascal/S

```
   0   var a:real;
   0  begin
   0   readln(a);
   3   writeln(a,' ',sin(a),' ',sqrt(a));
  16  end.

 identifiers          link  obj  typ  ref  nrm  lev  adr
      29 A              0    2    3    0  255    1    5

 blocks    last lpar psze vsze
     1      28    1    0    0
     2      29   28    5    6

 arrays    xtyp etyp eref  low high elsz sizw

 code :

   0  0 1    5, 27      2, 62       , 1 1    5, 29      2,
   5 24     32, 29      4, 1 1     5,  8     11, 29      2,
  10 24     32, 29      4, 1 1     5,  8     15, 29      2,
  15 63       , 31       ,

Start PCODE-Interpreter 3.1
```

Abb. 4.1.8
Ausschnitt aus einer Bildschirmausgabe; das Programm hat den Namen „TEST0“ erhalten

Das Grundprogramm findet die Position der EPROMs mit Hilfe der Bibliotheksfunktion automatisch, so daß man sonst nichts weiter beachten muß. Auch spielt es keine Rolle, wo sich das Grundprogramm befindet der Pascal-Compiler verwendet ausschließlich „TRAP"-Befehle für den Zugriff auf das Grundprogramm.

Testen wir nun den neu eingebauten Compiler. Dazu wird die Bibliotheksfunktion aufgerufen, und es erscheint *Abb. 4.1.4.*

Der Pascal-Compiler wurde gefunden. Die Startadresse gibt die absolute Position des Programms aus, je nachdem, wohin man den EPROM-Satz gesteckt hat. Wir starten den Compiler, indem wir die Taste J drücken. Er zeigt uns eine Fehlermeldung. Der Compiler sagt: „program incomplete". Das bedeutet „Programm unvollständig", klar, denn wir haben ja noch kein Programm eingegeben. Die Programmeingabe geschieht, wie beim Assembler, mit Hilfe des Editors. Bevor wir uns an das Programmieren in Pascal machen, geben wir ein kleines Testprogramm ein, um die Funktion endgültig zu prüfen. *Abb. 4.1.5* zeigt das Programm, so wie es per Editor einzugeben ist. Achtung, keine Punkte, Strichpunkte oder Kommas vergessen, der Compiler bemängelt sonst das Programm bei der Übersetzung!

Bitte wählen Sie nun im Optionen-Menue 80 Zeichen pro Zeile aus, damit auch alle Ausgaben auf dem Bildschirm sichtbar werden.

Dann wird der Compiler über die Bibliothek gestartet, und falls kein Eingabefehler vorliegt, meldet er sich nach der Übersetzung mit dem Text „Start PCODE-Interpreter 3.1", und der Cursor blinkt darunter. Gibt man daraufhin 3.1 ein, so sollte der Bildschirm wie *Abb. 4.1.6* aussehen.

Sollte der Pascal-Compiler einen Fehler melden, so muß man in den Editor zurück und den Fehler ausbessern. Ein Beispiel für einen Fehlerfall zeigt *Abb. 4.1.7.* Hinter dem Begriff „real" wurde der Strichpunkt vergessen. Der Compiler stellt diesen Fehler aber erst in der nächsten Zeile fest und markiert diese Stelle mit einem Pfeil. Dort steht die Zahl 15. Am Schluß des Lisitings wird vom Compiler eine Fehlerliste ausgegeben, links steht die Fehlernummer und rechts eine Interpretation des Fehlers, in diesem Fall „;".

Die Zahlen links neben der Eingabezeile geben die Anzahl der erzeugten „PCODE"-Befehle an. Man kann sich den erzeugten PCODE auch auf dem Bildschirm ausgeben, dazu muß man dem Programm den Namen „TEST0" geben. Die *Abb. 4.1.8* zeigt eine entsprechende Liste, die am Schluß des Programms ausgegeben wird. Neben dem PCODE auch interne Informationen über Variable etc. auf dem Bildschirm angezeigt.

Tritt während des Programmlaufs ein Fehler auf, so wird die mögliche Fehlerursache ausgegeben, wie auch eine aktuelle Belegung von Variablen etc. *Abb. 4.1.9* zeigt ein Beispiel.

Hier noch ein allgemeiner Hinweis: Es ist unmöglich, hier alle Pascal-Besonderheiten und Befehle genau zu behandeln, da die Sprache sehr umfangreich ist und es somit viele Einzelheiten gibt. Im Literaturverzeichnis finden Sie Bücher mit ausführlichen Pascal-Beschreibungen. In den folgenden Abschnitten werden wir anhand von praktischen Beispielen ein paar Besonderheiten von Pascal kennenlernen.

```
*************************
* PASCAL/S  Pcode-Compiler V3.1 *
* (C) 1984 Rolf-Dieter Klein    *
* Version nach PASCAL/S von     *
* N.Wirth 1976, E.T.H Zuerich   *
*************************

   0
   0  program fehlertest(input,output);
   0   var a:real;
   0  begin
   0   a:=0;
   4   writeln(1/a);
  10  end.

Start PCODE-Interpreter 3.1


 Abbruch bei PC= 8 wegen  Division durch 0

   A          =     0.000000000000e+00

▌

                                        F=Flip M=Menue
```

Abb. 4.1.9 Fehlermeldung „Division durch 0", die nach dem Start des Programms auftritt

Pflichtenheft

1. Anzeige: Höhe
2. Anzeige: Treibstoffmenge
3. Anzeige: Sinkgeschwindigkeit
4. Eingabe: Schub

Abb. 4.2.1 Das Pflichtenheft zur Mondlandung in Pascal

Abb. 4.2.2
Das Struktogramm zur Mondlandung

4.2 Mondlandung in Pascal

Aufgabe ist es, eine Mondfähre zu laden. Dabei soll die Ausgabe in Form von Zahlenwerten auf dem Bildschirm erscheinen. Als Eingabe wird der Antriebs-Schub angegeben. Um zunächst von einer eindeutigen Aufgabenstellung ausgehen zu können, halten wir unser Vorhaben in einem Pflichtenheft fest (*Abb. 4.2.1*).

Wie wir es bereits von unserer Assemblerprogrammierung her kennen, ist der nächste wichtige Schritt die Umsetzung in ein Struktogramm. Wie gehen wir dabei vor? Zunächst überlegt man sich einmal, wie das Programm im groben arbeiten soll. *Abb. 4.2.2* zeigt die erste Näherung. Das Programm ist in drei Teile geteilt. Der erste Teil legt alle Startwerte fest, also die Anfangshöhe und die Startgeschwindigkeit, sowie den Treibstoffvorrat. Dann folgt der mittlere Teil; dort wird ein Spielblock wiederholt, der die Berechnungen und Ausgaben durchführt. Dieser Spielblock wird so lange wiederholt, bis entweder eine Landung oder eine Bruchlandung erfolgte. Und im letzten Teil wird das Spielergebnis angezeigt.

Jeden dieser Blöcke kann man nun schrittweise weiter verfeinern. Man gelangt zu *Abb. 4.2.3* mit der höchsten Verfeinerungsstufe. Zunächst werden die Startwerte definiert. Dabei werden hier schon die Variablennamen verwendet, wie sie auch im Pascal-Programm vorkommen. Die „hoehe" wird mit „anfhoehe" belegt; „anfhoehe" ist dabei eine Konstante, die im Programm natürlich noch angegeben werden muß. Die Geschwindigkeit „geschw" soll 0 sein, und der Treibstoffvorrat „treibst" wird mit „anftreib" belegt.

Im Spielblock wird dann der Treibstoffvorrat abgefragt. Ist er kleiner als 0, dann wird er wieder mit 0 belegt, denn es gibt keinen negativen Treibstoffvorrat. Dann werden die aktuellen Werte der Variablen „hoehe", „geschw" und „treibst" auf den Bildschirm ausgegeben. Danach folgt die Eingabe des Schubes. Dabei sollte die Eingabe auf einen Wertebereich begrenzt werden, um nicht unmögliche Vorschubleistungen eingeben zu können. Hier wurde der Bereich –9 bis +9 zugelassen.

Wenn der Treibstoffvorrat größer als 0 ist, also „treibst>0", dann wird die Variable „schub" zum Inhalt von „geschw" addiert. Dabei muß man berücksichtigen, daß eine positive Geschwindigkeit die Fähre vom Mond wegbewegt. Danach wird ein Gravitationsanteil „grav" von der Geschwindigkeit subtrahiert und damit die Mondanziehung simuliert. Daraufhin ist die neue Höhe zu berechnen und schließlich der neue Treibstoffvorrat. Er nimmt bei jedem Schleifendurchlauf um einen konstanten Wert, hier 1, ab. Ferner wird die Betätigung von Steuerdüsen mit einem Anteil von 0.1 mal dem Betrag des Schubes angerechnet.

Der Spielblock wird so lange wiederholt, bis eine Höhe kleiner oder gleich 0 erreicht ist. Dann muß geprüft werden, ob es sich um eine gute Landung oder um eine Bruchlandung handelt. Wenn der Betrag der Höhe kleiner als 0.5 und der Betrag der Sinkgeschwindigkeit kleiner als 1 ist, soll es sich um eine gute Landung handeln, sonst ging die Fähre zu Bruch.

Das Struktogramm kann man nun in ein Programm umsetzen. Dabei spielt die verwendete Programmiersprache zunächst keine Rolle, man könnte das Programm auch in Assembler schreiben. Wir nehmen uns jetzt die Realisierung in Pascal vor. Durch die

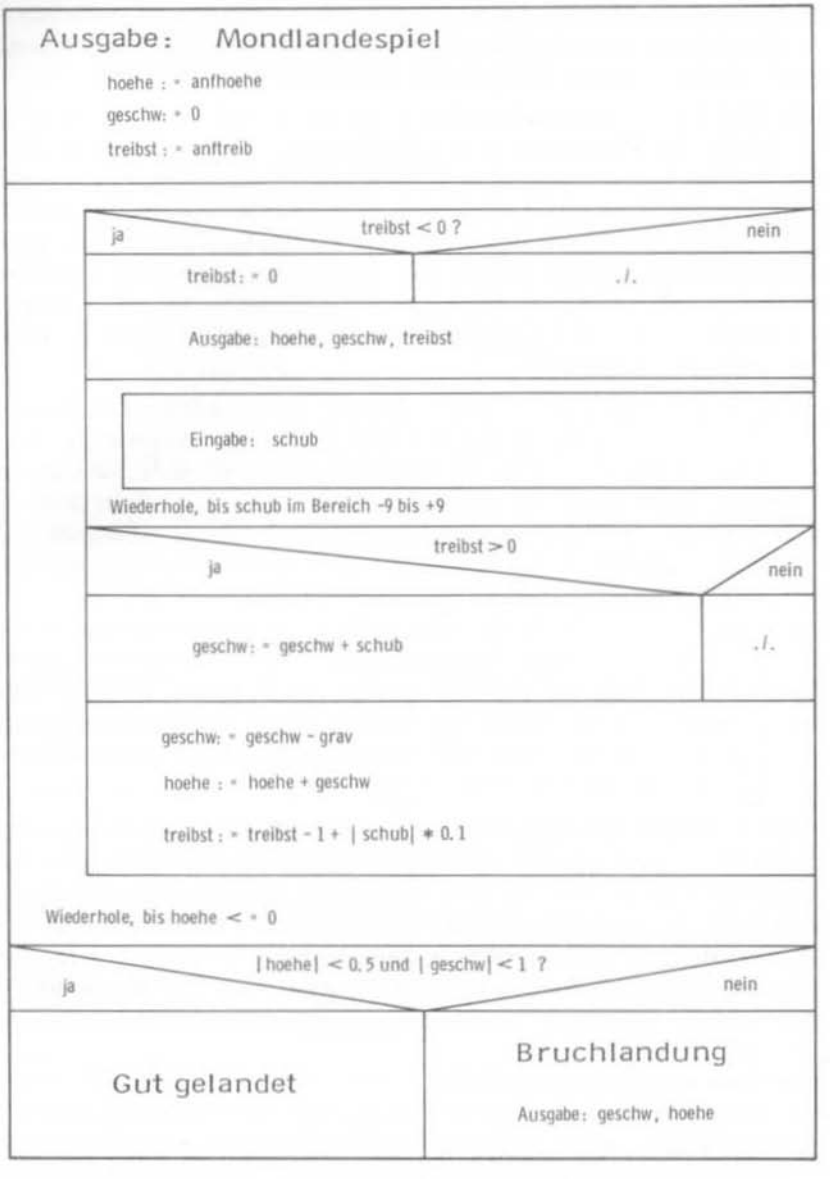


Abb. 4.2.3 Das Struktogramm zum Spielblock

Verwendung von Gleitkomma-Zahlen, also Zahlen, die Nachkommastellen oder Exponenten haben können, in unseren geplanten Berechnungen, ist Pascal ohnehin sehr gut geeignet.

Die *Abb. 4.2.4* zeigt das fertige Pascal-Programm. Es muß mit Hilfe des Editors eingegeben werden. Dann kann man die Bibliotheksfunktion aufrufen und dort den Pascal-Compiler starten. Der Pascal-Compiler schreibt an die linke Seite die Anzahl der erzeugten PCODE-Befehle. Gleichzeitig erscheint auch die Meldung „Start PCODE-Interpreter 3.1" und die erste Ausgabe des Programms. Das Mondlandespiel ist also bereits in Aktion. Gibt man einen falschen Wert, z. B. -10, ein, so wird die Anfrage einfach wiederholt, so wie es das Struktogramm vorschreibt.

Nach dem Ablauf des Spiels gelangt man durch die Eingabe von „M" zurück ins Grundprogramm. Will man das Programm erneut starten, so kann man es entweder neu übersetzen, oder man ruft den PCODE-Interpreter direkt auf. Dazu gibt es einen zweiten Eintrag in der Bibliothek (*Abb. 4.2.5*). Der PCODE-Interpreter startet dann das übersetzte Programm direkt. Dies funktioniert aber nur dann richtig, wenn das Programm zuvor korrekt übersetzt, und nicht durch einen RESET unterbrochen wurde.

Wie ist ein Pascal-Programm aufgebaut? Jedes Pascal-Programm beginnt mit dem Text „PROGRAM". Daran erkennt der Compiler den Anfang des Programms. Danach steht der Name des Programms. Dann folgen in Klammern zwei Angaben: INPUT sagt dem Compiler, daß in dem Programm Eingaben von der Tastatur vorkommen, und OUTPUT sagt dem Compiler, daß auch Ausgaben auf dem Bildschirm gemacht werden. Danach kommt ein Strichpunkt; dieser wird immer zur Trennung von Befehlen verwendet, denn in Pascal dürfen auch mehrere Befehle in einer Zeile stehen.

Jetzt folgt der Vereinbarungs-Baustein, auch Vereinbarungsteil genannt, der in Pascal immer direkt nach „program" folgen muß. Übrigens spielt die Groß- und Kleinschreibung in Pascal keine Rolle. Mit „const" wird die Konstantenvereinbarung eingeleitet. Diese entspricht im Assembler der „EQU"-Anweisung, um Namen anstelle von Zahlenwerten verwenden zu können.

Dann kommt der Variablendeklarationsteil, bei dem Speicherplatz für Variable reserviert wird. Die Variable „hoehe" bekommt einen Speicherplatz vom Typ „real". Das bedeutet, in der Variablen „hoehe" kann man eine Gleitkommazahl, also eine Zahl, die auch Nachkommastellen und einen Exponenten besitzen kann, speichern. Das ist nun neu gegenüber unserem Assembler. Der 68008 kann von sich aus nicht mit Gleitkommazahlen umgehen. Dazu befinden sich im PCODE-Interpreter umfangreiche Programmteile, die das verwirklichen. Es gibt in Pascal auch Integerzahlen, die einer 16Bit-Variablen im Assembler entsprechen. Die Gleitkommazahlen haben bei uns 12 Stellen (mit Schutzstellen 14) und einen Exponenten, der maximal 126 und minimal –126 betragen kann.

Nach dem Deklarationsteil folgt das Hauptprogramm. Das Hauptprogramm fängt immer mit „begin" an und hört mit „end." auf. Der Punkt nach end ist wichtig und darf nicht vergessen werden, sonst erscheint die Meldung „program incomplete".

Das Programm beginnt mit einer „writeln"-Anweisung. „writeln" heißt „WRITE WITH LINEFEED" und bedeutet, es soll eine Ausgabe gemacht und der Cursor anschließend in die nachfolgende Zeile gesetzt werden. In einfachen Anführungszeichen kann dann in Klammern gesetzt z. B. der Text stehen, der auszugeben ist.

```
program mondlandung(input,output);

{ Vereinbarungs - Baustein }

const   { Konstanten definieren }
 grav=1.6;
 anfhoehe=1000;
 anftreib=80;

var     { Variable definieren }
 hoehe   : real;
 treibst : real;
 geschw  : real;
 schub   : real;

{ Programm-Baustein }

begin
 writeln('Mondlandespiel');
 hoehe :=  anfhoehe;
 geschw  :=  0;
 treibst := anftreib;
 repeat
  if treibst <0 then treibst := 0;
  writeln('Hoehe:      ',hoehe:8:2,' m');
  writeln('Geschw.:    ',geschw:8:2,' m/s')
  writeln('Treibstoff: ',treibst:8:2);
  writeln;
  repeat
   write('Schub in m/s (-9..+9): ');
   read(schub);
   writeln;
  until abs(schub)<10;
  if treibst > 0
   then
     geschw:=geschw+schub;
  geschw:=geschw-grav;
  hoehe:=hoehe+geschw;
  treibst:=treibst-1-abs(schub)*0.1;
 until hoehe <=  0;
 if (abs(hoehe)<0.5) and (abs(geschw)<1)
  then
   writeln('Gut gelandet')
  else
   begin writeln('Bruchlandung mit');
     writeln(geschw:5:2,' m/s');
     writeln(hoehe:5:2,' m');
   end;
end.
```

Abb. 4.2.4 Das Programmlisting zur Mondlandung

```
Bibliothek

Name   : PCODE
Start  : 0900E4
Laenge : 008000

Starten J = JA
cr = weiter, M = Menue
```

Abb. 4.2.5 Direktes Starten des PCode-Interpreters

Mit dem Zeichen ":=" erfolgt eine Wertzuweisung an eine Variable, ähnlich wie im Grundprogramm. Dann folgt die Anweisung „repeat". Dazu gehört die Anweisung „until", die bei der PCODE-Adresse 92 steht. Die Zahlen, die auf der linken Seite stehen, geben ja die Anzahl der PCODE-Befehle, die erzeugt wurden, an.
„repeat until" realisiert eine Wiederholschleife, die dem „Wiederhole bis" exakt entspricht. Der Spielblock soll nämlich so lange wiederholt werden, bis die Höhe der Mondlandefähre kleiner oder gleich 0 ist. Mit der Anweisung „if" kann man eine Verzweigung realisieren. Wenn noch ein „else" folgt, so bedeutet das, daß der ELSE-Teil durchlaufen wird, wenn die Bedingung hinter „if" nicht erfüllt war. Im Programm gibt es auch noch eine Eingabeanweisung, nämlich „read", mit der der Wert der Variablen „schub" eingelesen wird. Bei der Ausgabe von Werten mit „writeln" ist eine Ausgabeformatierung möglich. Bei 122 z.B. wird die Höhe mit insgesamt 5 Stellen und 2 Nachkommastellen ausgeben. Die Zahl hinter dem Doppelpunkt nach der Variablen gibt die Gesamtzahl der Stellen an und die nächste Zahl die Zahl der Nachkommastellen. Läßt man sie beide weg, so wird die Zahl im Exponentialformat ausgegeben. In Zeile 97 gibt es noch eine interessante Funktion, die noch nicht bekannt ist. Mit abs( ) kann man den Absolutbetrag bilden.
Nun wäre es ganz schön, wenn man die Mondlandefähre auch noch mit Hilfe einer graphischen Darstellung auf dem Bildschirm einblenden könnte. Die Sprache Pascal bietet dazu leider keine direkten Möglichkeiten, doch gibt es eine Besonderheit in unserem Grundprogramm, die es uns erlaubt, die Fähre darzustellen. Bisher haben wir solche Darstellungen meistens mit unserer Schildkrötengrafik vorgenommen. Wir haben also fertige Unterprogramme im Grundprogramm aufgerufen. Es sollte daher möglich sein, von unserem Pascalprogramm aus diese Routinen aufzurufen und, das ist sehr wichtig, auch Werte an sie zu übergeben. Genau das wollen wir tun. Wir verwenden dafür einen Pascalbefehl, den wir bereits kennen: writeln. Wenn wir per writeln das Zeichen mit dem Code 1, also ein Zeichen, das gar nicht darstellbar ist, zum Bildschirm senden, so erkennt das Grundprogramm daran unser Vorhaben. Alle folgenden Zeichen bis zum nächsten Zeilenvorschub (der am Ende von writeln automatisch erfolgt) werden jetzt nicht zum Bildschirm, sondern in einen Zwischenspeicher transportiert. Der für dieses Umleiten zum Zwischenspeicher abschließende Zeilenvorschub löst gleichzeitig im Grundprogramm die Analyse des Zwischenspeicherinhalts aus. Steht an der ersten Stelle in diesem Speicher ein „E", so bedeutet dies, führe das folgende Unterprogramm aus. Bevor die ganze Geschichte zu abstrakt wird, sehen wir uns den ganzen writeln-Befehl an:

```
writeln(chr(1),'E @SCHREITE 100');
```

Mit chr() wird der Code des in Klammern stehenden Wertes ausgeben, es wird also der Code 1, der das Umleiten zum Zwischenspeicher bewirkt, losgeschickt. Der Text zwischen den beiden einfachen Anführungszeichen landet dann in diesem Zwischenspeicher und wird vom Grundprogramm analysiert. E steht für execute, also führe aus, und zwar das dahinter angegebene Unterprogramm. Dies ist der uns bereits bekannte Schreitebefehl. Die 100 dahinter soll dabei als Parameter übergeben werden und die Festlegung der Schrittweite bewirken.
Abb. 4.2.6 zeigt ein kleines Programmbeispiel dazu. Wenn man das Programm übersetzt und startet, so erscheint ein 100 Schritte langer Strich auf dem Bildschirm. Man kann

```
program schildkroete(input,output);

begin
 writeln(chr(1),'E @SCHREITE 100');
end.
```

Abb. 4.2.6 Aufruf der Unterprogramme aus dem Grundprogramm

```
Textstart=009000 Fenster=009000 Tor=009000      amer CTRL-J=Hilfe
```

```
program schildkroete(input,output);
var i:integer;

begin
 for i:=1 to 36 do begin
  writeln(chr(1),'E @SCHREITE 10');
  writeln(chr(1),'E @DREHE 10');
 end;
end.
```

```
Textstart=009000 Fenster=009000 Tor=009000      amer CTRL-J=Hilfe
```

Abb. 4.2.7 Ein Kreisprogramm in Pascal

```
program gitter(input,output);
var x,y:integer;

begin
 for x:=0 to 9 do begin
  writeln(chr(1),'E @MOVETO 0 ',x*50,' 0');
  writeln(chr(1),'E @DRAWTO 0 ',x*50,' 225');
 end;
 for y:=0 to 9 do begin
  writeln(chr(1),'E @MOVETO 0 0 ',y*25);
  writeln(chr(1),'E @DRAWTO 0 450 ',y*25);
 end;
end.
```

```
Textstart=009000 Fenster=009000 Tor=009000      amer CTRL-J=Hilfe
```

Abb. 4.2.8 Ein Gitter-Programm

damit ganz interessante Versuche durchführen, so z. B. *Abb. 4.2.7*, das Programm für einen Kreis.

Man kann auch mehr als nur einen Parameter beim Aufruf an das Unterprogramm übergeben. Dabei werden die angegebenen Parameter nacheinander den Registern D0, D1, D2 ... bis maximal D7 übergeben, und zwar als Langwort.

Abb. 4.2.8 zeigt ein Beispiel für mehrere Parameter. Dabei werden die Unterprogramme „MOVETO" und „DRAWTO" verwendet. Sie verlangen die x-Koordinate in Register D1 und die y-Koordinate in Register D2. Register D0 wird nicht verwendet. Beim Aufruf muß man es aber dennoch belegen, da immer bei Register D0 angefangen wird, die Parameter ans Grundprogramm zu übertragen. Hier wurde einfach der Wert 0 an das Register D0 übergeben. Wie man sieht, können nicht nur konstante Werte, sondern auch Variable an das Unterprogramm übergeben werden; man muß dabei darauf achten, daß immer ein Leerzeichen zwischen den einzelnen Zahlen steht. Die Zusammenstellung im writeln-Befehl kann man einfach testen, indem man zunächst den Befehl „chr(1)," wegläßt und damit eine Ausgabe auf den Bildschirm erhält. Durch diesen Test spart man sich viel Mühe bei der Fehlersuche in eigenen Programmen.

Übrigens gibt es im Grundprogramm noch weitere Möglichkeiten, die eine Kommunikation mit Pascal erlauben. Den „E"-Befehl kennen Sie ja schon, es gibt ferner noch den „P"-Befehl und einen „G"-Befehl.

„P" steht für PUT, damit kann man Werte zu einer Adresse des Speichers oder der Peripherie schicken. Beispiel:

```
WRITELN(chr(1),'P $A000 7');
```

legt den Wert 7, als Bytegröße auf der Adresse $A000 ab, oder:

```
WRITELN(chr(1),'P $FFFFFF30 $55');
```

gibt den Wert $55 am Port mit der Adresse $50 aus.

„G" ist die Abkürzung von GET. Damit kann man Speicherinhalte oder Portinhalte lesen. Dabei kann man den Wert mit einer nachfolgenden READ-Anweisung einlesen, die dazu eine Integer-Zahl lesen muß. Das Echo auf dem Bildschirm wird dabei automatisch unterdrückt. Beispiel:

```
WRITELN(chr(1),'G $8000');
READ(I);
```

In der Variablen I (vom Typ Integer) ist dannach der Inhalt der Speicherzelle $8000 enthalten. Dabei wird immer eine Bytegröße verarbeitet. Weiteres Beispiel:

```
WRITELN(chr(1),'G $FFFFFF30');
READ(I);
```

Diesmal wird der am Port $30 anstehende Wert eingelesen und kann dann mit der Variablen I weiterverarbeitet werden.

Zurück zur Mondlandung:

Abb. 4.2.9 zeigt die Assemblerliste der Unterprogramme, die für das Mondlandespiel gebraucht werden. Wir werden diese Unterprogramme in der soeben behandelten Weise von unserem Pascalprogramm aus aufrufen. Mit Hilfe des FIGUR-Befehls wird die Mondlandefähre dargestellt. *Abb. 4.2.10* zeigt das Pascalprogramm mit den Aufrufen. Zum Start gibt man zuerst das Assemblerprogramm ein und übersetzt es. Dann legt man den Pascalteil auf eine neue Editor-Adresse (Textstart wählen), oder man überschreibt das alte Programm und startet den Pascal-Compiler. Die *Abb.4.2.11* zeigt eine mögliche

```
0E9C00
0E9C00                    *
0E9C00                    * ASSEMBLER HILFSROUTINEN FUER PASCAL-MONDLANDUNG
0E9C00                    *
0E9C00
00A000                     ORG $A000    * Z.B. DORT ABLEGEN
00A000
00A000                    TAB1:
00A000  08 02 09 00 00    DC.B 8,2,9,0,0,7,3,2,2,4,4,4,4,6,6,5,1,0,0,10
00A005  07 03 02 02 04
00A00A  04 04 04 06 06
00A00F  05 01 00 00 0A
00A014                    TAB2:
00A014  02 09 00 00 07    DC.B 2,9,0,0,7,3,2,2,4,4,4,4,6,6,5,1,0,0,10
00A019  03 02 02 04 04
00A01E  04 04 06 06 05
00A023  01 00 00 0A
00A027
00A027  00                DS.B 0
00A028
00A028  0000A014          TABVAR: DC.L TAB2
00A02C
00A02C                    ASSFAEHRE:
00A02C  2079 0000A028      MOVEA.L TABVAR,A0
00A032  3401               MOVE D1,D2
00A034  3200               MOVE D0,D1
00A036  303C 0003          MOVE #$3,D0
00A03A  4EB9 000E328C      JSR @FIGUR
00A040  4E75               RTS
00A042
00A042                    ASSAUS:
00A042  23FC 0000A000      MOVE.L #TAB1,TABVAR
00A048  0000A028
00A04C  4E75               RTS
00A04E
00A04E
00A04E                    ASSAN:
00A04E  23FC 0000A014      MOVE.L #TAB2,TABVAR
00A054  0000A028
00A058  4E75               RTS
00A05A
00A05A
00A05A                    TABMOND:
00A05A  0002 0005 0000     DC.W  2,5,0,0,7,2,5,4,3,9,0,0,0,3,0,0,0,0,0
00A060  0000 0007 0002
00A066  0005 0004 0003
00A06C  0009 0000 0000
00A072  0000 0003 0000
00A078  0000 0000 0000
00A07E  0000
00A080
00A080                    ASSMOND:
```

Abb. 4.2.9 Assembler-Unterprogramme für die Mondlandung mit graphischer Darstellung

```
00A080  303C 0001        MOVE #1,D0
00A084  4241             CLR D1
00A086  4EB9 000E0E0C    JSR @NEWPAGE
00A08C  4EB9 0000A0A6    JSR ASS1
00A092  303C 0000        MOVE #0,D0
00A096  4241             CLR D1
00A098  4EB9 000E0E0C    JSR @NEWPAGE
00A09E  4EB9 0000A0A6    JSR ASS1
00A0A4  4E75             RTS
00A0A6
00A0A6
00A0A6
00A0A6                  ASS1:
00A0A6  323C 0190        MOVE #400,D1
00A0AA  4242             CLR D2
00A0AC  4EB9 000E0EE2    JSR @MOVETO
00A0B2  41F9 0000A05A    LEA TABMOND,A0
00A0B8                  AK1:
00A0B8  3418             MOVE (A0)+,D2
00A0BA  4EB9 000E0F4C    JSR @DRAWTO
00A0C0  0641 000A        ADD #10,D1
00A0C4  0C41 01FE        CMP #510,D1
00A0C8  6600 FFEE        BNE AK1
00A0CC  4E75             RTS
00A0CE
00A0CE                  ASSKRATER:
00A0CE  303C 000C        MOVE #12,D0
00A0D2  4EB9 000E0D66    JSR @CMD
00A0D8                  WA1:
00A0D8  4EB9 000E0E38    JSR @SYNC
00A0DE  6700 FFF8        BEQ WA1
00A0E2                  WA2:
00A0E2  4EB9 000E0E38    JSR @SYNC
00A0E8  6700 FFF8        BEQ WA2
00A0EC  4EB9 000E3FEC    JSR @CLRSCREEN          zu Abb. 4.2.9
00A0F2  4E75             RTS
00A0F4
00A0F4

0E8E86  Ende-Symboltabelle
```

```
**********************************
* PASCAL/S  Pcode-Compiler V3.1 *
* (C) 1984 Rolf-Dieter Klein    *
* Version nach PASCAL/S von     *
* N.Wirth 1976, E.T.H Zuerich   *
**********************************

    0  program mondlandung(input,output);
    0
    0  const
    0   grav=1.6;
    0   anfhoehe=1000;
    0   anftreib=80;                          zu Abb. 4.2.10
```

```
  0
  0  var
  0   x,y     : integer;
  0   hoehe   : real;
  0   treibst : real;
  0   geschw  : real;
  0   schub   : real;
  0
  0  procedure faehre(x,y:integer);
  0  begin
  0   writeln(chr(1),'E assfaehre ',x,' ',y)
 11  end;
 12
 12  procedure krater;
 13  begin
 13   writeln(chr(1),'E asskrater')
 18  end;
 19
 19  procedure mond;
 20  begin
 20   writeln(chr(1),'E assmond')
 25  end;
 26
 26  procedure triebaus;
 27  begin
 27   writeln(chr(1),'E assaus');
 33  end;
 33
 33  procedure trieban;
 34  begin
 34   writeln(chr(1),'E assan')
 39  end;
 40
 40
 40  begin
 41   writeln(chr(1),'E @clrscreen');
 47   writeln('Mondlandespiel');
 50   mond;
 52   trieban;
 54   hoehe :=  anfhoehe;
 58   geschw  :=  0;
 62   treibst := anftreib;
 66   repeat
 66    if treibst<0 then treibst:=0;
 75    x:=480;
 78    y:=trunc(hoehe/4);
 85    faehre(x,y);
 89    writeln('Hoehe:       ',hoehe:8:2,' m');
 98    writeln('Geschw.:     ',geschw:8:2,' m/s');
107    writeln('Treibstoff: ',treibst:8:2);
114    writeln;
115    repeat
```

Abb. 4.2.10 Das Pascal-Programm für die Mondlandung mit graphischer Darstellung

```
PASCAL/S Pcode-Compiler V3.1  (C) 1984

  115     write('Schub in m/s (-9..+9): ');
  117     read(schub);
  119     writeln;
  120    until abs(schub)<10;
  126    if treibst > 0
  128     then
  131       geschw:=geschw+schub
  133     else
  136       triebaus;
  139    geschw:=geschw-grav;
  144    hoehe:=hoehe+geschw;
  149    treibst:=treibst-1-abs(schub)*0.1;
  160   until hoehe <=  0;
  165   if (abs(hoehe)<0.5) and (abs(geschw)<1)
  174    then
  176     writeln('Gut gelandet')
  178    else
  179     begin
  180      krater;
  182       writeln('Bruchlandung mit');
  185       writeln(geschw:5:2,' m/s');
  192       writeln(hoehe:5:2,' m');
  199     end;
  199  end.
```

zu Abb. 4.2.10

```
Hoehe:       548.20 m
Geschw.:     -25.20 m/s
Treibstoff:   51.20

Schub in m/s (-9..+9): 0

Hoehe:       521.40 m
Geschw.:     -26.80 m/s
Treibstoff:   50.20

Schub in m/s (-9..+9): 0

Hoehe:       493.00 m
Geschw.:     -28.40 m/s
Treibstoff:   49.20

Schub in m/s (-9..+9): 0

Hoehe:       463.00 m
Geschw.:     -30.00 m/s
Treibstoff:   48.20

Schub in m/s (-9..+9): ▌
```

Abb. 4.2.11
Während der Landephase

Situation auf dem Bildschirm beim Spiel in der neuen Variante. Wenn Namen, die in Pascal aufgerufen werden, nicht definiert sind, z. B. weil sie falsch geschrieben sind, so werden die Unterprogramme einfach nicht ausgeführt, der Pascallauf wird aber nicht abgebrochen. Damit kann man umgekehrt zunächst einmal das Pascalprogramm entwickeln und erst später die Assemblerteile hinzufügen.

In unserem Programm nach Abb. 4.2.10 gibt es noch etwas Neues. In Pascal kann man Unterprogramme mit PROCEDURE definieren, dies geschieht nach dem Vereinbarungsblock des Hauptprogramms und vor dem Hauptprograterprogramme mit PROCEDURE definieren, dies geschieht nach dem Vereinbarungsblock des Hauptprogramms und vor dem Hauptprogramm. Prozeduren können auch Parameter besitzen, wie z.B. FAEHRE. Die Parameter werden in Klammern angeben, zuzüglich einer Typdeklaration, hier INTEGER. Diese Unterprogramme mit Parameterübergaben gestalten Programme übersichtlich und sollten darum häufig verwendet werden. Die Parameter werden hier an das Assemblerunterprogramm ASSFAEHRE weitergereicht.

4.3 Die Türme von Hanoi

In diesem Abschnitt werden wir die Technik der Rekursion kennenlernen. Dazu ist besonders eine Programmieraufgabe geeignet, die aber nicht ganz einfach zu lösen ist: die Türme von Hanoi.

Bei diesem Spiel gibt es drei Stäbe, auf die man Scheiben mit unterschiedlichem Durchmesser stecken kann. *Abb. 4.3.1* zeigt das Schema. Auf dem Stab 1 sind die Scheiben, der Größe nach geordnet, aufgebracht. Ziel ist es, die Scheiben in möglichst wenig Zügen auf den Stab 3 zu transportieren. Dabei sind ein paar Regeln zu beachten.

1. Es darf immer nur eine Scheibe gezogen werden.
2. Es muß immer eine kleinere Scheibe auf einer größeren zu liegen kommen.

Man beginnt dann bei dem Stab 1 und legt die obere Scheibe auf Stab 2, dann legt man die nächste Scheibe von Stab 1 auf Stab 3 und nun die Scheibe von Stab 2 auf Stab 3 usw. Der Computer soll hier den günstigsten Weg finden und durchführen.

Bevor wir die Lösung dieses Problems verstehen können, beginnen wir mit der Lösung eines einfacheren Problems, der Berechnung der Fakultät eines Wertes, die auch rekursiv erfolgen kann. Die Fakultät ist wie folgt definiert:

Fakultät von 2 ist $1 \cdot 2 = 2$
Fakultät von 3 ist $1 \cdot 2 \cdot 3 = 6$
Fakultät von 4 ist $1 \cdot 2 \cdot 3 \cdot 4 = 24$ usw.

Man kann die Fakultät auch noch genauer definieren, sie wird übrigens durch ein nachgestelltes Ausrufezeichen gekennzeichnet:

$n! = n \cdot (n-1)!$
und es gilt
$0! = 1$

Die Formel bedeutet folgendes: Fakultät von n ist n multipliziert mit der Fakultät von n minus 1. Also z. B. gilt:

$3! = 3 \cdot 2!$

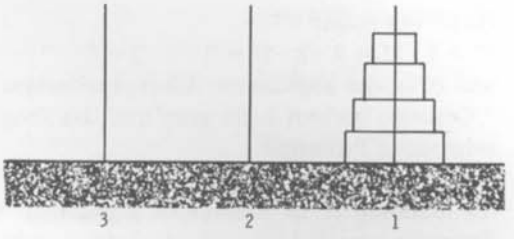


Abb. 4.3.1
Die Türme von Hanoi

```
program fakultaet(input,output);

var i:integer;
    r:real;

function fac(a:real):real;
 begin
  if a=0
   then fac:=1.0
   else fac:=fac(a-1)*a
 end;

begin
 repeat
  writeln;
  write('fac von: ');
  read(i);
  writeln;
  r:=fac(i);
  writeln(' ist ',r);
  writeln(' ist ',r:14:0)
 until false
end.
```

Abb. 4.3.2
Ein Programm zur Berechnung der Fakultät

```
Textstart=009000 Fenster=009000 Tor=009002      amer CTRL-J=Hilfe
```

```
fac von: 3

 ist     6.000000000000e+00
 ist           6.

fac von: 10

 ist     3.628800000000e+06
 ist       3628800.

fac von: 69

 ist     1.711224524267e+98
 ist 171122452426700000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000.

fac von: 84

 ist     3.314240134530e+126
 ist 331424013453000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000.

fac von:
```

Abb. 4.3.3
Beispiele auf dem Bildschirm

Damit gilt dann:
$3! = 3 \cdot 2! = 3 \cdot 2 \cdot 1! = 3 \cdot 2 \cdot 1 \cdot 0!$
und 0! ist das sogenannte Abbruchkriterium, und wurde mit 1 definiert.

Genauso einfach kann man nun das Programm realisieren (*Abb. 4.3.2*). *Abb. 4.3.3* zeigt einen Probelauf.

Im Programm wurde mit FUNCTION die Funktion fac definiert. Als Parameter wird die Variable A in Klammern angegeben. Nun steht im Programmteil einfach die Formulierung, wie sie in der Definition steht. Wenn A=0 ist, also 0! berechnet werden soll, so erhält fac den Wert 1, das ist dann der Wert, den die Funktion an das aufrufende Programm übergibt. War A nicht 0, so wird die rekursive Formel gebraucht, und fac ergibt sich zu fac von A minus 1 mal A. Man hätte auch A·(FAC(A-1)) schreiben können, die Wirkung ist die Gleiche. Hier ruft sich die Funktion FAC aber selbst auf, man sagt dazu rekursiv. Damit beim neuen Durchlauf nicht der gleiche Speicherplatz für die Variable A verwendet wird, legt Pascal die Variablen auf einen Stack und beim neuen Aufruf wird einfach ein neuer Bereich im Stack verwendet. Nur deshalb kann man eine Funktion überhaupt rekursiv verwenden.

Aufgabe:

Schreiben Sie Funktion FAC, ohne eine Rekursion zu verwenden.
Die Lösung der Türme von Hanoi ist ebenfalls rekursiv. *Abb. 4.3.4* zeigt das Programm. Das Programm ist erstaunlich kurz, dennoch nicht einfach zu verstehen. *Abb. 4.3.5* zeigt einen Probeablauf mit 3 Scheiben.

Bei der Lösung von rekursiven Problemen, sind zwei Kriterien zu definieren. Ein Abbruchkriterium, das ist hier gegeben, wenn die Turmhöhe kleiner oder gleich 0 ist, und die Rekursion für einen allgemeinen Fall. Es gilt also, eine komplexe Aufgabe zu einem einfachen Fall zu reduzieren.

Verfolgen Sie bitte mit Papier und Bleistift den Ablauf des Programms nach Abb. 4.3.4 mit einer Turmhöhe von 2. Bei jedem erneuten Aufruf der Prozedur müssen alle Parameter der jeweiligen Prozedur gemerkt werden.

Wenn man das Programm verstanden hat, kann man sich an eine komfortable Gestaltung der Ausgabe machen. *Abb. 4.3.6* zeigt die Assemblerunterprogramme für eine graphische Darstellung der Scheiben und *Abb. 4.3.7* das dazugehörige Pascalprogramm. Die Scheiben werden dann nach dem Start auf dem Bildschirm so bewegt, wie in der Aufgabenstellung verlangt. *Abb. 4.3.8* zeigt den Stand während des Ablaufs. Man kann bei dem Programm nicht beliebige Turmhöhen angeben, es würden dann nicht mehr alle Scheiben übereinander passen.

Aufgaben:

1. Erweitern Sie das Programm so, daß auch die Dicke der Scheiben bei der Ausgabe berücksichtigt wird.
2. Erweitern Sie das Programm auf große Turmhöhen.

```
program hanoi(input,output);
 var total:integer;

procedure schiebe(hoehe,von,nach,mit : integer);

begin
 if hoehe > 0
  then begin
   schiebe(hoehe-1,von,mit,nach);
   writeln('von ',von,' nach ',nach);
   schiebe(hoehe-1,mit,nach,von);
  end;
 end;

 begin
  repeat
   writeln;
   write('Turmhoehe ');
   read(total);
   writeln;
   schiebe(total,1,3,2);
  until false
end.
```

Abb. 4.3.4
Programm: Die Türme von Hanoi

```
Start PCODE-Interpreter 3.1

Turmhoehe 3

von 1 nach 3
von 1 nach 2
von 3 nach 2
von 1 nach 3
von 2 nach 1
von 2 nach 3
von 1 nach 3

Turmhoehe
```

Abb. 4.3.5
Bildschirmausgabe für eine Turmhöhe von 3

```
Textstart=009000 Fenster=009000 Tor=00900E     amer CTRL-J=Hilfe
```

```
      0E9C00                    *
      0E9C00                    * ASSEMBLERUNTERPROGRAMME
      0E9C00                    * FUER DIE TUERME VON HANOI
      0E9C00                    *
      00A800                    ORG $A800       * FREIEN PLATZ NEHMEN
      00A800
      00A800                    ASSINIT:        * BILDSCHIRM VORBEREITEN
      00A800  4EB9 000E3FEC      JSR @CLRSCREEN
      00A806  4240               CLR D0
      00A808  4241               CLR D1
      00A80A  4EB9 000E1F06      JSR @SETFLIP
      00A810  4E75               RTS
      00A812
      00A812                    ASSPLATTE:      * D0= HOEHE, D1=X , D2= Y
      00A812  41F9 0000A828      LEA SCHEIBE,A0 * SCHEIBE AUSGEBEN
      00A818  4EB9 000E328C      JSR @FIGUR
      00A81E  4E75              RTS
      00A820
      00A820                    ASSFEST:        * SCHEIBE FIXIEREN
      00A820  4EB9 000E3286      JSR @SETFIG
      00A826  4E75               RTS
      00A828
      00A828                    SCHEIBE:        * SCHEIBENFORM
      00A828  00 00 02 04 04     DC.B 0,0,2,4,4,4,4,6,0,0,10
      00A82D  04 04 06 00 00
      00A832  0A
      00A833
      00A833
      00A833                    END

      0E8A96  Ende-Symboltabelle
```

Abb. 4.3.6 Assembler-Unterprogramme für die Türme von Hanoi mit graphischer Ausgabe

```
**********************************
* PASCAL/S  Pcode-Compiler V3.1 *
* (C) 1984 Rolf-Dieter Klein    *
* Version nach PASCAL/S von     *
* N.Wirth 1976, E.T.H Zuerich   *
**********************************

    0  program hanoi(input,output);
    0
    0   const
    0    x1 = 200;   (* Lage der Tuerme in x-Richtung *)
    0
    0    y0 = 40;    (* Hoehe der Tuerme *)
    0    yoben = 100; (* Hoehe max der Tuerme *)
    0
    0
    0   var total:integer;
    0   x,y: integer;
    0
    0   h: array[1..3] of integer;  (* hoehe *)
    0
    0
    0  procedure setturm; (* Startturm *)
    0  var i:integer;
    0
    0  begin
    0   for i:=1 to total do begin
    4    writeln(chr(1),'E assplatte ',(1+total-i)*2,' ',x1,' ',y0+i*14);
   30    writeln(chr(1),'E assfest');
   36   end;
   37   writeln(chr(27),'=',chr(32+23),chr(32+31),' 1        2        3');
   55   h[1]:=total;
   60   h[2]:=0;
   65   h[3]:=0;
   70  end;
   70
   70  procedure platte(hoehe,von,nach:integer);
   71
   71  var x,y,i:integer;
   71   dx:integer;
   71   xziel:integer;
   71
   71  begin
   71   for y:=h[von]*14+y0 to yoben do
   82    writeln(chr(1),'E assplatte ',hoehe*3,' ',x1+50*(von-1),' ',y);
  107   h[von]:=h[von]-1;
  117   x:=x1+50*(von-1);
  126   xziel:=x1+50*(nach-1);
  135   if xziel < x then dx:=-1 else dx:=1;
  147   repeat
  147    writeln(chr(1),'E assplatte ',hoehe*3,' ',x,' ',yoben);
  165    x:=x+dx;
  170   until x=xziel;
  174   h[nach]:=h[nach]+1;
```

Abb. 4.3.7 Pascal-Programm für die Türme von Hanoi mit graphischer Ausgabe

```
184   for y:=yoben downto h[nach]*14+y0 do
195    writeln(chr(1),'E assplatte ',hoehe*3,' ',x1+50*(nach-1),' ',y);
220   writeln(chr(1),'E assfest');
226  end;
226
226
226
226
226
226  procedure schiebe(hoehe,von,nach,mit:integer);
```

```
227  begin
227   if hoehe > 0
229    then begin
231     schiebe(hoehe-1,von,mit,nach);
240     writeln(chr(27),'=  ');
246     writeln('von ',von,' nach ',nach);
255     platte(hoehe,von,nach);
261     schiebe(hoehe-1,mit,nach,von);
270    end;
270   end;
270
270   begin
271    repeat
271     writeln;
272     write('Turmhoehe ');
274     read(total);
276     writeln(chr(1),'E assinit');
282     setturm;
284     writeln;
285     schiebe(total,1,3,2);
291    until false
291  end.
```

zu Abb. 4.3.7

---

von 3 nach 2

Abb. 4.3.8 Das Programm in Aktion

4.4 Taschenrechner selbstgebaut

Ziel dieses Abschnittes ist es, mit Syntaxdiagrammen vertraut zu werden und sie als Programmierhilfsmittel verwenden zu können.

Wir wollen mit unserem Rechner einen Taschenrechner simulieren. In der Praxis benötigt man solche Taschenrechnerprogramme sehr häufig, um gelegentlich kleine Rechnungen mit dem Computer durchführen zu können. Man spart sich dabei den Griff zum üblichen Taschenrechner und verwendet den genaueren und schnelleren Computer. Die *Abb. 4.4.1* zeigt das Pflichtenheft. Neben den vier Grundrechenarten soll der Taschenrechner auch die Klammertechnik beherrschen. Außerdem die Rechenregeln, also Punkt vor Strich und Klammer vor allen anderen Operationen.

Für die Entwicklung verwenden wir hier nicht unser Struktogramm, sondern wir arbeiten mit einem Syntaxdiagramm. Diese Methode ist für das Lösen rekursiver Probleme, wie in diesem Fall, besser geeignet. *Abb. 4.4.2* zeigt die Elemente eines Syntaxdiagramms. Es gibt eine Flußrichtung, die uns durch das Diagramm führt. Außerdem gibt es Terminalsymbole, die beim Auftreten erkannt werden müssen. Und es gibt den Beschreibungsaufruf, der wieder aus einem Syntaxdiagramm besteht.

Zum besseren Verständnis sehen wir uns ein Syntaxdiagramm an, das einen Pascal-Namen definiert (*Abb. 4.4.3*). Ein Name beginnt grundsätzlich mit einem Buchstaben und besteht aus mindestes einem Buchstaben. Es können nach dem ersten Buchstaben weitere Buchstaben oder Ziffern folgen: HUBER, MEYER, MUELLER, B5HHY sind alles gültige Namen. In dem Namen dürfen aber keine anderen Zeichen stehen als die angegebenen. Es ist also noch festzustellen, was Buchstaben und was Ziffern sind. Wie wir bereits gehört haben, stellen die Beschreibungsaufrufe wiederum Syntaxdiagramme dar. So ist in der *Abb. 4.4.4* festgelegt, was Buchstaben sind, und in *Abb. 4.4.5* finden wir die Definition für die Ziffern.

Mit Syntaxdiagrammen kann man sehr anschaulich Sprachen beschreiben. So ist Pascal zum Beispiel vollständig mit Syntaxdiagrammen beschrieben worden. *Abb. 4.4.6* zeigt die Definition der IF-Anweisung, so wie sie in Pascal erlaubt ist. Dabei beginnt die Anweisung mit dem Terminalsymbol IF. Danach muß ein Ausdruck folgen, der die Bedingung berechnet. Dann folgt das Symbol THEN, und danach kann eine Anweisung stehen. An dieser Stelle ist das Syntaxdiagramm schon rekursiv, denn die IF-Anweisung ist ebenfalls eine Anweisung. Folglich kann man nach THEN wieder eine IF-Anweisung stellen. Danach kann optional ein ELSE stehen, dann folgt eine weitere Anweisung, der ELSE-Teil.

So kann man alle Befehle von Pascal definieren. Es ist dann für den Benutzer einfach nachzuschauen, ob eine Programmkonstruktion erlaubt ist oder nicht. Genauso einfach ist es aber auch für den Programmierer des Compilers, diesen fehlerarm zu gestalten, denn er kann sich am Syntaxdiagramm orientieren und es fast 1 : 1 in ein Programm umsetzen. Noch dazu kann man fast alle Syntaxdiagramme getrennt umsetzen, ohne sich weiter um die Beschreibungsaufrufe zu kümmern. Diese werden dann wiederum gesondert behandelt. In dieser Art wollen wir auch bei unserem Taschenrechnerprogramm vorgehen.

Abb. 4.4.7 zeigt die Definition eines Ausdrucks, wie ihn der Taschenrechner verstehen soll. Ein Ausdruck kann dabei wiederum ein Term sein, der ein Vorzeichen haben darf, gefolgt von einem weiteren Term usw.

Abb. 4.4.1 Pflichtenheft: Taschenrechner

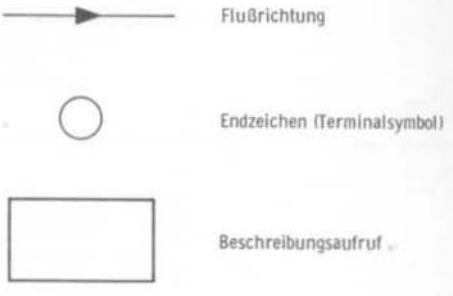


Abb. 4.4.2
Die Elemente von Syntaxdiagrammen

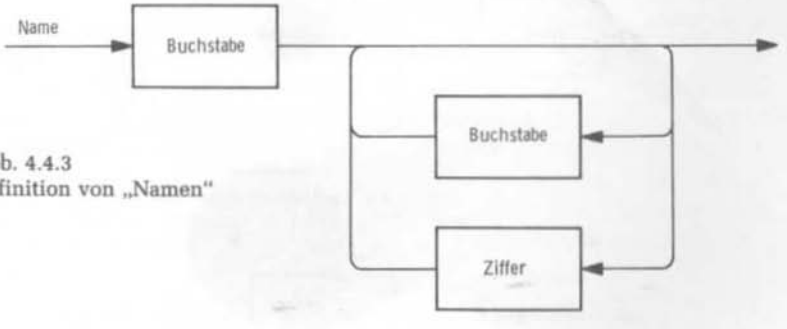


Abb. 4.4.3
Definition von „Namen"

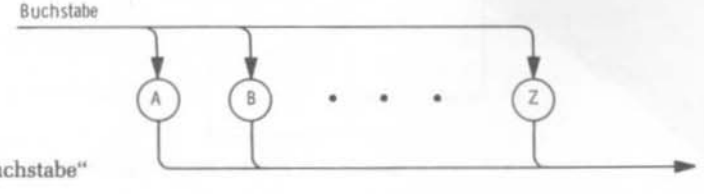


Abb. 4.4.4
Definition von „Buchstabe"

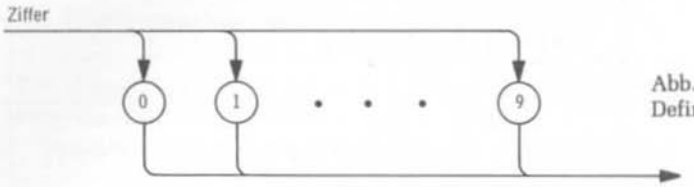


Abb. 4.4.5
Definition von „Ziffer"

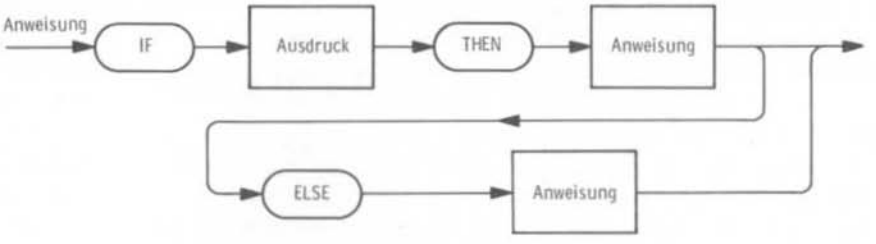


Abb. 4.4.6 Eine „Anweisung" in Pascal

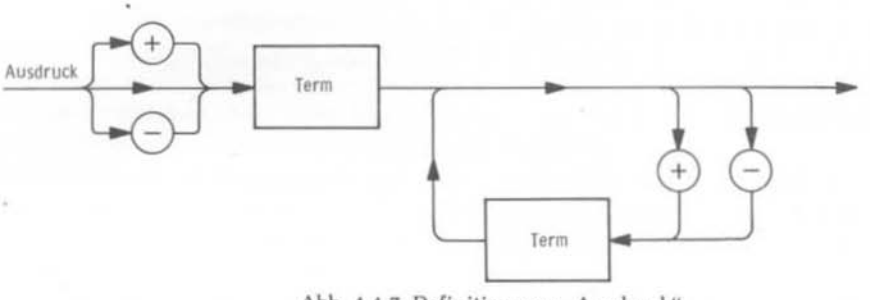


Abb. 4.4.7 Definition von „Ausdruck"

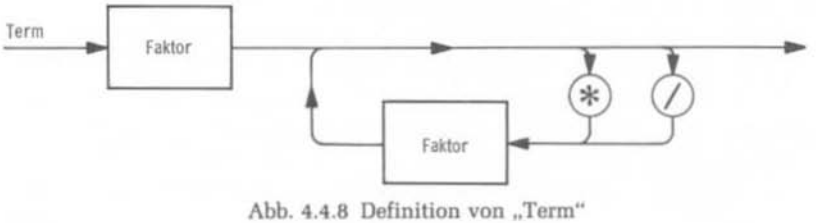


Abb. 4.4.8 Definition von „Term"

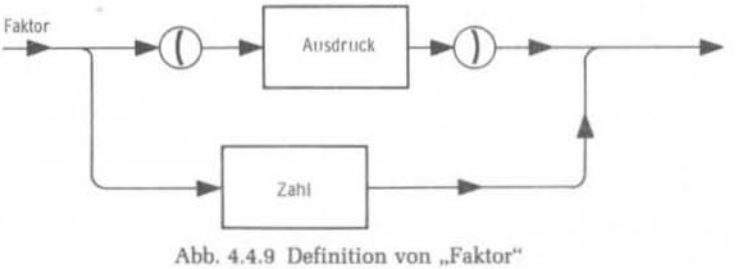


Abb. 4.4.9 Definition von „Faktor"

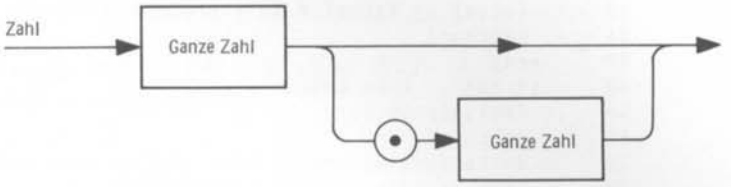


Abb. 4.4.10
Definition von „Zahl"

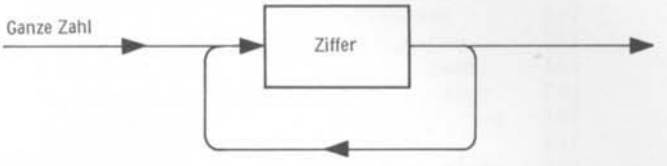


Abb. 4.4.11
Definition von „Ganze Zahl"

```
**********************************
* PASCAL/S  Pcode-Compiler V3.1 *
* (C) 1984 Rolf-Dieter Klein    *
* Version nach PASCAL/S von     *
* N.Wirth 1976, E.T.H Zuerich   *
**********************************

    0  program taschenrechner(input,output);
    0
    0  { Vereinbarungs-Baustin }
    0
    0  var
    0   ch : char;
    0   a : real;
    0
    0  { Unterprogramm-Baustein }
    0
    0  procedure getchar;  { Zeichen einlesen }
    0   begin
    0    read(ch)
    2   end;
    2
    2  procedure expr(var exprval : real);  { Ausdruck }
    3   var nexttermval : real;
    3    ch1 : char;
    3
    3   procedure term(var termval : real);  { Term }
    3   var nextfacval : real;
    3    ch2 : char;
    3
    3   procedure factor(var facval :real);  { Faktor }
    3    var fac1val : real;
    3   begin                                          zu Abb. 4.4.12
    3    if ch='(' then begin
    7     getchar;
   10     expr(facval);
   14     if ch = ')'
   16      then getchar
   18      else writeln(' Fehler ) fehlt ')
```

```
  24    end else begin
  26     facval:=0;
  30     while (ord(ch)>=ord('0')) and (ord(ch)<=ord('9')) do begin
  42      facval := facval * 10 + ord(ch) - ord('0');
  56      getchar;
  59     end;
  60     if ch='.' then begin
  64      fac1val:=10.0;
  67      getchar;
  70      while (ord(ch)>=ord('0')) and (ord(ch)<=ord('9')) do begin
  82       facval:=facval+(ord(ch)-ord('0'))/fac1val;
  94       fac1val:=fac1val*10;
 100       getchar;
 103      end;
 104     end;
 104    end;
 104   end;
 104
 104   begin (* term *)
 105    factor(termval);
 108    while (ch='*') or (ch='/') do begin
 116     ch2 := ch;
 119     getchar;
 122     factor(nextfacval);
 125     if ch2='*'
 127      then termval:=termval*nextfacval

PASCAL/S Pcode-Compiler V3.1  (C) 1984

 131      else termval:=termval/nextfacval
 137    end;
 141   end;
 141
 141   begin (* expr *)
 142    ch1 := ch;
 145    if (ch='+') or (ch='-') then getchar;
 156    term(exprval) ;
 159    if ch1='-' then exprval:=-exprval;
 167    while (ch='+') or (ch='-') do begin
 175     ch1 := ch;
 178     getchar;
 181     term(nexttermval);
 184     if ch1 = '+'
 186      then exprval := exprval + nexttermval
 190      else exprval := exprval - nexttermval
 196    end
 199   end;
 200
 200  { Programm - Baustein }
 200
 200  begin
 201   writeln('Taschenrechner') ;
 204   repeat
 204    write(':');  { Bereitmeldung }
 206    getchar;     { Erstes Zeichen lesen }
 208    expr(a);     { dann Ausdruck auswerten }
 211    writeln;     { Zeilenvorschub }
 212    writeln(' = ',a);       { Ergebnis ausgeben }
 217    writeln(' = ',a:14:8);  { auf verschiedene Art }
 224   until false   { unbegrenzt wiederholen }
 224  end.
```

Abb. 4.4.12
Das Taschenrechner-Programm

```
Start PCODE-Interpreter 3.1
Taschenrechner
:3.5+7.2

 =      1.070000000000e+01
 =     10.70000000
:7-9.1

 =     -2.100000000000e+00
 =     -2.10000000
:2*3.1

 =      6.200000000000e+00
 =      6.20000000
:10/3

 =      3.333333333333e+00
 =      3.33333333
:-4+3-6*2/8

 =     -2.500000000000e+00
 =     -2.50000000
:3*(4-5)

 =     -3.000000000000e+00
 =     -3.00000000
:(3+5+6*(9-1))/(4+5.1)*(5-7)

 =     -1.230769230769e+01
 =    -12.30769230
:

 =      0.000000000000e+00
 =      0.00000000
:

 =      0.000000000000e+00
 =      0.00000000
:
```

Abb. 4.4.13 Beispiele für Testeingaben

Beispiel:
-TERM
TERM1+TERM2-TERM3

Ähnlich sieht die Definition eines Terms aus. *Abb. 4.4.8* zeigt das Syntaxdiagramm. In Faktor, die Definition zeigt *Abb. 4.4.9*, wird dann wiederum die Definition von Ausdruck verwendet. Hier finden wir also wieder unsere Rekursion. Finden all diese Definitionen keine Anwendung für den Faktor, so ist dieser eine Zahl.

Damit sind die vier Grundrechenarten und die Klammerrechnung definiert. Wo aber sind die Rechenregeln? Sie stecken in der Anordung der Syntaxdiagramme. Denn bei einem Ausdruck werden zunächst die Terme ausgewertet und dann zuletzt addiert oder subtrahiert, es wird also Punkt vor Strich gerechnet. Sind dann in einzelnen Faktoren Klammerausdrücke, so werden auch diese zunächst vollständig berechnet. Zu definieren sind jetzt nur noch „Zahl" und dafür wiederum „ganze Zahl". Dies geschieht in den *Abb. 4.4.10* und *4.4.11*.

Die Umsetzung der Syntaxdiagramme kann man am fertigen Pascal-Listing sehen; es ist in der *Abb. 4.4.12* abgedruckt. *Abb. 4.4.13* zeigt ein paar Eingabebeispiele dazu.

Bei der Realisation gibt es eine Besonderheit. Es werden zunächst die Prozeduren EXPR, TERM und FACTOR angeben, und dann folgt der jeweilige Programmteil in umgekehrter Reihenfolge. Dies ist in Pascal nötig, da man Programme nur dann aufrufen darf, wenn man sie schon definiert hat. Bei rekursiven Programmen, die sich gegenseitig aufrufen, kann das zu Problemen führen, und man muß sie ineinander schachteln, um die Schwierigkeit zu umgehen.

Hier sei nochmals auf die Pascal-Literatur verwiesen, siehe Literaturverzeichnis im Anhang.

Aufgaben:

1. Welchen einfachen Fehler enthält die Programmrealisation im Programmteil FAKTOR? Vergleichen Sie dazu Syntaxdiagramm und Programmstruktur. Zeichnen Sie das Syntaxdiagramm so, daß es dem Programm entspricht!
2. Welches Ergebnis liefert der Taschenrechner bei der Eingabe „3+."? (Siehe auch Aufgabe 1!)
3. Erweitern Sie das Syntaxdiagramm für die Eingabe von Gleitkommazahlen, die auch einen Exponenten haben dürfen, und programmieren Sie das Syntaxdiagramm!
4. Erweitern Sie den Taschenrechner um trigonometrische Funktionen!
5. Erweitern Sie den Taschenrechner um eine Speicherfunktion. Dazu sollen Variable A bis Z verwendet werden können. Zum Setzen einer Variablen soll man z.B. =A schreiben dürfen. Konstruieren Sie zuerst das Syntaxdiagramm!

5 Gosi – Ein Compiler für den 68000/68008

Nachdem wir im letzten Kapitel die Sprache Pascal kennengelernt haben, wollen wir uns hier mit der Sprache Gosi beschäftigen. Gosi ist die Abkürzung für „Graphisch Orientierte Sprache I". Die Sprachelemente sind größtenteils der Sprache Logo entlehnt. Gosi enthällt aber nicht alle Sprachelemente der Sprache Logo, sondern nur Teile daraus. Dabei sind sowohl der deutsche Sprachschatz (IWT-Logo) als auch die englische Variante (MIT-Logo) mit eingebaut. Die deutsche Variante ist jedoch immer voreingestellt. Gosi enthält auch ein paar Elemente, die in Logo nicht vorkommen, insbesondere zur Verknüpfung mit Assembler-Programmen für Aufgaben der Steuerung und Regelung. Auf Abweichungen wird im Text entsprechend hingewiesen. Gosi ist eine Compilersprache. Dieser Compiler übersetzt die Gosi-Programme in Assembler-Programme für den 68000/68008.

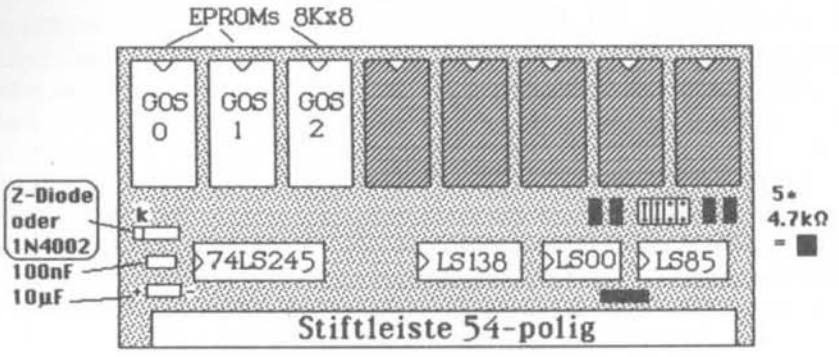
5.1 Einbau von Gosi

Gosi wird in Form dreier EPROMs geliefert. Gosi ist voll verschiebbar, es kann also an irgendeiner Stelle im Hauptspeicher installiert werden. Zum Einbau verwendet man eine getrennte ROA64 Baugruppe. Die Anordnung der EPROMs kann z. B. wie in *Abb. 5.1.1* erfolgen; die Baugruppe wurde hier auf Adresse $C0000 gelegt. Also nochmals, die genaue Lage der EPROMs, die Gosi enthalten, spielt keine Rolle, sie müssen nur zusammenhängend und mit aufsteigender Nummerierung eingesetzt werden.

Gosi wird, wie wir es von Pascal bereits kennen, über die Bibliotheks-Funktion aufgerufen. Der Arbeitsspeicher wird durch Gosi automatisch festgelegt.

Starten Sie nun einmal den Gosi-Compiler, indem Sie die Taste „J" drücken, nachdem Gosi in der Bibliothek erscheint. Wie zu sehen ist, benötigt Gosi ein eingegebenes Programm. Dieses muß zuerst mit dem Editor eingegeben werden.

Nun nochmals zur Arbeitsspeicheraufteilung: Gosi benötigt mehrere Speicherbereiche, die aber automatisch verwaltet werden. Auch Konflitke zwischen einzelnen Bereichen werden von Gosi erkannt und als Fehler gemeldet. Die *Abb. 5.1.2* zeigt eine schematische Darstellung. Wenn, wie in diesem Fall, das Grundprogramm auf Adresse 0 liegt, dann beginnt der RAM-Speicher bei der Adresse $8000. Dort befindet sich ein fester Bereich, den das Grundprogramm als Arbeitsspeicher verwendet. Direkt dahinter ist die Symboltabelle des Grundprogramms angeordnet, die einen unterschiedlich großen Raum einnimmt. Wie wir bereits wissen, hängt dies von der Anzahl der definieren Symbole ab. Der Textbereich sei z. B. ab Adresse $9000 eingestellt. Die Textlänge hängt dann natürlich vom eingegebenen Text ab. Am Ende des Speichers liegt dann noch der Stack, der ebenfalls vom Grundprogramm verwendet wird und in



z.B. Brücken 16,17 eingesetzt,
Brücken 18,19 offen.

Abb. 5.1.1 Einbau der Gosi-EPROMs

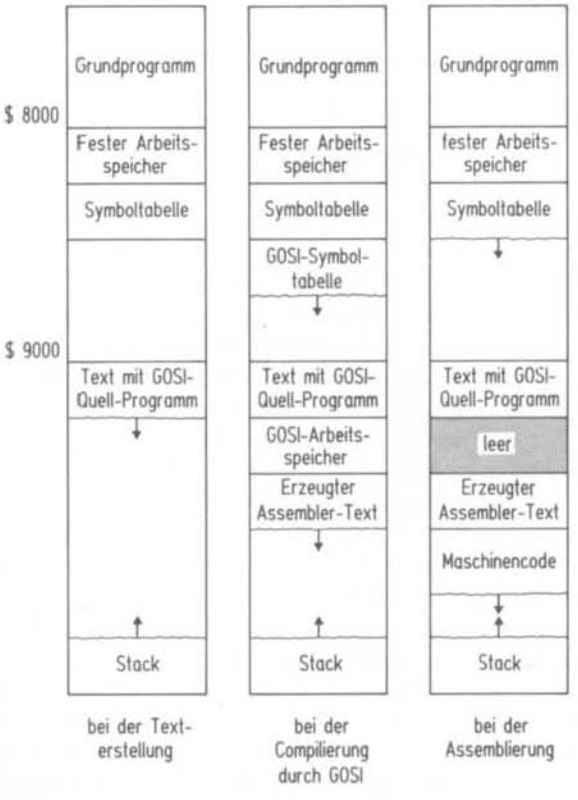


Abb. 5.1.2 Die Speicheraufteilung unter Gosi

Richtung niederer Adressen wächst. Auch hier könnten Konflikte auftreten. So kann sich die Symboltabelle mit dem Text überschneiden. Dies führt dann zu Datenverlust. Der Assembler kann in solchen Fällen auch Fehlermeldungen wie ADDRESSERROR oder multiple definied symbols ausgeben. Wie wir sehen, ist es sinnvoll, die Lage der Symboltabelle und des Textes immer genau zu prüfen, das Grundprogramm tut es nicht.

Wird nun Gosi aufgerufen, so werden weitere Bereiche reserviert. Zunächst muß Gosi die Quelle in einen Assemblertext umsetzen. Dazu wird eine Gosi-Symboltabelle angelegt, die unmittelbar auf die Symboltabelle des Grundprogramms folgt. Da die Symboltabelle des Grundprogramms während des Übersetzungsvorgangs nicht weiter wachsen kann und die Textquelle ebenfalls festgelegt ist, ist der Bereich dahinter für die Zeit der Übersetzung frei verwendbar. Hier wird der Gosi-Arbeitsspeicher angelegt. Dahinter schließlich wird der durch Gosi erzeugte Assemblertext abgespeichert. Wenn die Gosi-Symboltabelle mit dem Text zu kollidieren droht, so meldet dies der Gosi-Compiler durch eine Fehlermeldung. Ebenfalls wird geprüft, ob der erzeugte Assemblercode in den Stack-Bereich läuft; auch dafür hält Gosi die eindeutige Meldung „erzeugter Code laeuft in Stackgebiet" bereit.

Eine andere Möglichkeit besteht darin, daß der Assembler-Text in ein nicht existierendes Speichergebiet abgelegt wird, z. B. wenn der Stack im unteren RAM-Bereich liegt, der Text aber in einer getrennten ROA-Baugruppe übersetzt wird. Auch dieser Fall wird automatisch geprüft und durch eine Fehlermeldung angezeigt.

Nachdem der Assemblertext erzeugt wurde, wird von Gosi automatisch der Assembler aufgerufen, allerdings nur dann, wenn kein Fehler aufgetreten ist. Der Assembler erzeugt dann den Maschinencode, der direkt hinter dem Assemblertext abgelegt wird. Da die Symboltabelle jetzt wieder vom Grundprogramm verwaltet wird, kann es vorkommen, daß der Gosi-Text beim Wachsen der Symboltabelle überschrieben wird.

Nach erfolgreicher Übersetzung durch den Assembler wird anschließend das Gosi-Programm gestartet. Möchte man den Assemblercode getrennt bearbeiten, so kann man dies tun, wenn man als alten Textstart „ASSTXT" angibt. Man muß jedoch dann vor einer Übersetzung selbst mit einer ORG-Anweisung einen freien Platz für den Maschinencode wählen. Will man Gosi im Einzelschritt testen, so muß man den Assembler getrennt aufrufen, denn durch den Aufruf des Gosi-Compilers wird der Debug-Mode automatisch ausgeschaltet. Für den Einzelschritt muß ebenfalls eine ORG-Anweisung in die Assemblerquelle eingebaut werden.

All das klingt komplizierter als es ist. Im nächsten Abschnitt probieren wir einfach mal ein paar Beispiele aus.

5.2 Einfache Sprachelemente

Die Elemente der Schildkrötensprache kennen Sie ja schon aus den ersten Kapiteln. In Gosi gibt es dafür direkte Sprachelemente. Die *Abb. 5.2.1* zeigt ein einfaches Programm.

Tippen Sie dieses Programm einmal ein, und rufen Sie dann den Gosi-Compiler auf. Die Übersetzung geht sehr schnell. Kurz nachdem die Quelle nochmals auf den Bildschirm gegeben wurde, erscheint die Ausgabe des Assemblers (*Abb. 5.2.2*). Dieses Assemblerprogramm wurde von dem Gosi-Compiler erzeugt. Darin erkennen Sie die

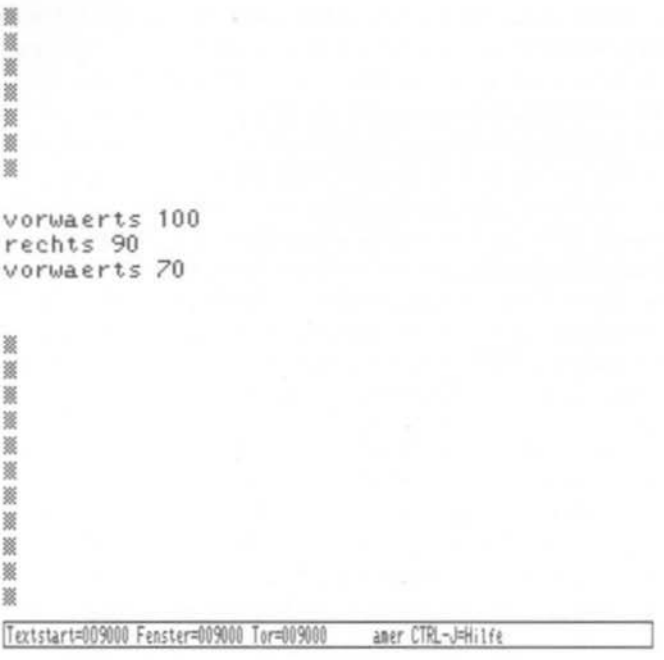


Abb. 5.2.1 Ein einfaches Gosi-Programm

```
0092AE                  ##########################
0092AE                  # GOSI68K COMPILER VERS 3.0 #
0092AE                  # (C) 1985, MUENCHEN       #
0092AE                  # ROLF-DIETER KLEIN        #
0092AE                  ##########################
0092AE                  GOSI:
0092AE  4EB9 000E3168    JSR @SETR5
0092B4  4EB9 000E3FEC    JSR @CLRSCREEN
0092BA  4EB9 000E1342    JSR @FIRSTTIME
0092C0  7064             MOVEQ.L #$64,D0
0092C2  4EB9 000E1288    JSR @SCHREITE
0092C8  203C FFFFFFA6    MOVE.L #-$5A,D0
0092CE  4EB9 000E12F6    JSR @DREHE
0092D4  7046             MOVEQ.L #$46,D0
0092D6  4EB9 000E1288    JSR @SCHREITE
0092DC  4E75             RTS
0092DE
0092DE                  # ENDE GOSI-UEBERSETZUNG
0092DE                   END

0E8B80  Ende-Symboltabelle
```

Abb. 5.2.2
Der erzeugte Assembler-Code

VORWAERTS	VW
RUECKWAERTS	RW
LINKS	LI
RECHTS	RE
STIFTHOCH	SH
STIFTAB	SA
SCHR16TEL	

Tabelle 5.2.1
Die ersten
Schildkröten-Befehle

Elemente der Zeichensprache wieder. Das Programm beginnt mit der Marke „Gosi"; unter diesem Namen kann man es auch nach dem ersten Lauf direkt vom Grundprogramm aus starten. Am Anfang stehen noch ein paar allgemeine Aufrufe. „SETA5" sorgt dafür, daß das Register A5 mit dem Start des RAM-Bereichs geladen wird. Manchmal greift Gosi nämlich direkt auf die Speicherzellen in dem reservierten RAM-Bereich des Grundprogramms zurück. Dann wird „CLRSCREEN" aufgerufen, um den Bildschirm zu löschen und die Zeichenausgabe vorzubereiten. Mit „FIRSTTIME" wird garantiert, daß die Schildkröte in der Bildmitte startet. Dann erfolgt ein SCHREITE-Befehl mit $64, also 100 Schritten, ein DREHE-Befehl, der mit einem negativen Wert geladen wird – die Schildkröte sollte sich also nach rechts drehen – und dann wieder der SCHREITE-Befehl, diesmal mit 70 Schritten, also sedezimal $46. Hier beginnt der Maschinencode bei der Adresse ($92AE), die Gosi als erste freie Speicherzelle hinter dem Assembler-Text feststellen konnte. Mit „RTS" wird das Programm beendet.

Nachdem die Übersetzung durch den Assembler beendet ist, wird das Programm gestartet, und die entsprechenden Linien werden auf den Bildschirm gezeichnet. In der linken oberen Ecke sieht man den Cursor, der bei Gosi immer eingeschaltet wird. Will man ihn nicht sehen, so muß man einen Befehl in Gosi programmieren, der ein Unterprogramm des Grundprogramms aufruft, doch dazu später.

Tab. 5.2.1 zeigt eine Zusammenstellung der Schildkröten-Sprachelemente von Gosi. Manche Befehle kann man auch abkürzen, die Abkürzungen sind rechts daneben aufgelistet. Alle Befehle können sowohl in Groß- als auch in Kleinschreibung angegeben werden.

VORWAERTS, oder kurz VW, läßt die Schildkröte in Blickrichtung laufen. Die Anzahl der Schritte wird dazu hinter den Befehl geschrieben. Man muß immer mindestens ein Leerzeichen zwischen Befehle und Zahlenwerte setzen.
Beispiele:
VW100 ist falsch,
VW 100 ist richtig.

Der Befehl „RUECKWAERTS", oder „RW" läßt die Schildkröte entgegen der Blickrichtung laufen. Die Anzahl der Schritte wird ebenfalls als Parameter angegeben.

Mit „LINKS" oder „LI" dreht sich die Schildkröte nach links, dabei wird der Winkel in Grad angegeben.

„RECHTS" oder „RE" dreht die Schildkröte nach rechts, der Winkel wird ebenfalls in Grad angegeben.
„STIFTHOCH" oder „SH" hebt den Schreibstift, und „STIFTAB" oder „SA" senkt ihn wieder. Mit „SCHR16TEL" kann die Schildkröte um 1/16 Bildpunkt in Blickrichtung schreiten, dazu wird der Wert als Parameter angegeben.

Ein Programmbeispiel zum Test dieser Befehle zeigt *Abb. 5.2.3* und *Abb. 5.2.4* die dabei erzeugte Zeichnung.

In Gosi ist es möglich, mehrere Befehle in eine Zeile zu schreiben. Dadurch ergibt sich eine recht kompakte Darstellung. Man sollte das aber nicht übertreiben, denn sonst werden die Programme schnell unübersichtlich. Um Schleifen programmieren zu können, benötigt man noch einen weiteren Befehl: Die WIEDERHOLE-Anweisung:
WIEDERHOLE oder kurz: WH.

```
vorwaerts 40 rechts 90
vw 30 re 45
rueckwaerts 10 links 45
rw 20 li 90
stifthoch vw 20 stiftab vw 20
sh vw 20 sa vw 20
re 90
schr16tel 100
```

Abb. 5.2.3 Ein komplexeres Beispiel

```
Textstart=009000 Fenster=009000 Tor=009000      amer CTRL-J=Hilfe
```

WIEDERHOLE anzahl [befehle]

WH anzahl [befehle]

Abb. 5.2.5
Der Wiederhole-Befehl

```
wh 36 [ wh 36 [ vw 10 re 10 ] re 10 ]
```

Abb. 5.2.6
Der verschachtelte
Wiederhole-Befehl

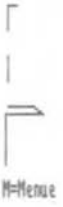


Abb. 5.2.4 Die
Bildschirmausgabe

```
Textstart=009000 Fenster=009000 Tor=009000      amer CTRL-J=Hilfe
```

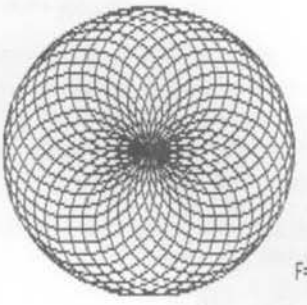


Abb. 5.2.7
Selbst einfache Anweisungen führen zu interessanten Bildern

```
GOSI68K Compiler Vers 3.0  (C) 1984 Rolf-Dieter Klein
*
*wh 36 [ wh 36 [ vw 10 re 10 ] re 10 ]
*
*
*--------------
Rolf-D.Klein 68000/08 Assembler 4.3 (C) 1984, Seite 1
009336                      ******************************
009336                      * GOSI68K COMPILER VERS 3.0 *
009336                      * (C) 1985, MUENCHEN        *
009336                      * ROLF-DIETER KLEIN         *
009336                      ******************************
009336                      GOSI:
009336  4EB9 000E3168        JSR @SETA5
00933C  4EB9 000E3FEC        JSR @CLRSCREEN
009342  4EB9 000E1342        JSR @FIRSTTIME
009348  7023                 MOVEQ.L #$23,D0
00934A                      L1:
00934A  2F00                 MOVE.L D0,-(A7)
00934C  7023                 MOVEQ.L #$23,D0
00934E                      L2:
00934E  2F00                 MOVE.L D0,-(A7)
009350  700A                 MOVEQ.L #$A,D0
009352  4EB9 000E1288        JSR @SCHREITE
009358  203C FFFFFFF6        MOVE.L #-$A,D0
00935E  4EB9 000E12F6        JSR @DREHE
009364  201F                 MOVE.L (A7)+,D0
009366  51C8 FFE6           DBRA D0,L2
00936A  203C FFFFFFF6        MOVE.L #-$A,D0
009370  4EB9 000E12F6        JSR @DREHE
009376  201F                 MOVE.L (A7)+,D0
009378  51C8 FFD0           DBRA D0,L1
00937C  4E75                 RTS
00937E
00937E                      * ENDE GOSI-UEBERSETZUNG
00937E                       END

0E8B80  Ende-Symboltabelle
```

Abb. 5.2.8
Die Ausgabe des Gosi-Compilers

```
sh aufxy 100 100
sa aufx 200
aufy 200
aufx 100
aufy 100
aufxy 200 200
aufkurs 0
vw 100
```

MITTE
AUFX
AUFY
AUFXY
AUFKURS AK

Tabelle 5.2.2 Befehle für absolute Positionierung

LINIE

```
Textstart=009000 Fenster=009000 Tor=009000      amer CTRL-J=Hilfe
```

Abb. 5.2.9 Die Positionsbefehle als Beispiel

Abb. 5.2.10 Die Ausgabe auf dem Bildschirm

```
inie 0 100 511 100
:h aufxy 150 200 sa
ih 4 [ vw 100 re 90 ]
'w 100 re 30
ih 4 [ vw 100 re 120 ]
:h aufxy 170 200 sa
ik 90
'w 60 re 90 vw 30 re 90 vw 60
:h mitte
'i
```

Abb. 5.2.11 Ein kombiniertes Beispiel

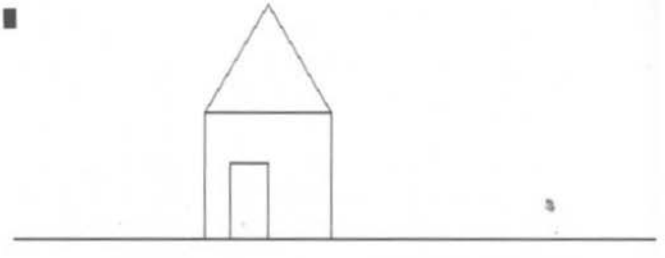


Abb. 5.2.12 So kann man schon einfache Zeichnungen erstellen

Die Syntax ist nicht ganz so einfach wie bei den bereits besprochenen Schildkröten-Befehlen. Hinter dem WIEDERHOLE-Befehl steht die Anzahl der Wiederholungen, *Abb. 5.2.5* zeigt ein Schema. Dann folgt ein „eckige Klammer auf", dahinter können beliebige weitere Gosi-Befehle stehen, auch weitere Wiederholungen. Beendet wird das Ganze durch ein „eckige Klammer zu".

Ein Programmbeispiel zeigt *Abb. 5.2.6*. Das Ergebnis auf dem Bildschirm ist in *Abb. 5.2.7* zu sehen. *Abb. 5.2.8* zeigt die gesamte Ausgabe des Gosi-Compilers. Am Anfang steht das Quell-Programm nochmals zur Kontrolle, und dann folgt die Ausgabe des Assemblers. Wenn man die Ausgabe nicht ständig auf dem Bildschirm wünscht, so kann man im Optionen-Menue durch Auswahl von „1=Assembler Optionen" die Ausgabe auf „Nur Fehlerausgabe" umschalten und erhält dann nur noch die Kopfzeile des Assemblers kurz auf dem Bildschirm. Im Optionen-Menue kann man auch die Ausgabe auf den Drucker umlenken, so wie es für Abb. 5.2.8 geschah. Das Grundprogramm liegt hier nicht auf Adresse 0, darum erscheint am Schluß des Listings die Adresse $e8b80. Lassen Sie sich dadurch also nicht verunsichern. Die Adresse hängt von der Lage der EPROMs und des RAMs ab und ist bei unterschiedlichen Konfigurationen unterschiedlich.

Tab. 5.2.2 zeigt weitere Gosi-Befehle. Mit dem Befehl „MITTE" wird die Schildkröte wieder in die Startlage gebracht. War dabei der Schreibstift gesenkt, so wie dies nach dem Einschalten oder nach der Ausführung des Befehls „SA" der Fall ist, so wird eine Linie von der augenblicklichen Position der Schildkröte zur Mitte gezeichnet. Mit dem Befehl „AUFX" kann man die x- und mit „AUFY" die y-Koordinate neu setzen. „AUFXY" setzt beide Koordinaten gleichzeitig neu. Dazu erhält AUFXY zwei Parameter als Angabe. Mit „AUFKURS" oder „AK" kann man die Blickrichtung der Schildkröte neu einstellen. Der Wert 90 steht dabei für 90°, also die Blickrichtung nach oben, und 0 für die Blickrichtung nach rechts.

Abb. 5.2.9 zeigt ein Programmbeispiel mit diesen Befehlen, *Abb. 5.2.10* die Bildschirmausgabe dazu.

Für die Angabe der x- und y-Koordinaten gilt der Wertebereich 0 bis 511. Außerhalb dieses Bereichs ist die Schildkröte nicht mehr sichtbar.

Beim „LINIE"-Befehl ist das anders. Er arbeitet mit den internen Befehlen MOVETO und DRAWTO, daher sind für die x-Koordinate der Bereich 0 bis 511 und für die y-Koordinate der Bereich 0 bis 255 zulässig. Der „LINIE"-Befehl verlangt vier Parameter:
LINIE x1 y1 x2 y2

Der Punkt (x1,y1) ist der Startpunkt, und der Punkt (x2,y2) ist der Endpunkt der Linie. Der LINIE-Befehl beeinflußt die Position der Schildkröte nicht. Wen das Bildschirmseiten-Umschalten stört, der kann es durch Aufruf von „VI" abschalten. Mit „ZI" wird das Umschalten wieder angestellt. „ZI" läßt auch die Schildkröte wieder sichtbar werden. Ein kombiniertes Programm zeigt *Abb. 5.2.11*, in *Abb. 5.2.12* ist das entstehende Haus zu sehen.

Hier sieht man schon die Vorteile der Sprache Gosi gegenüber der Assemblersprache; man kann Programme viel kompakter schreiben.

5.3 Prozeduren

Interessant wird es erst, wenn man Unterprogramme verwenden kann. In Gosi ist es natürlich möglich, Unterprogramme zu schreiben, man nennt sie Prozeduren. Ähnlich wie in Pascal genügt es dann später, sie im Hauptprogramm einfach durch ihren Namen aufzurufen.

Eine Unterprogramm-Definition wird in Gosi durch den Befehl „LERNE" eingeleitet. Die Definition wird durch einen weiteren Befehl, nämlich „ENDE" angeschlossen. Alle Befehle, die zwischen „LERNE" und „ENDE" stehen, stellen also das Unterprogramm dar. Unmittelbar hinter „LERNE" kann man einen Namen angeben. Dieser Name darf allerdings nicht identisch mit einem der Gosi-Befehle sein. Nach der Definition kann man das Unterprogramm an jeder beliebigen Stelle im Hauptprogramm durch diesen Namen aufrufen.

Ein Beispiel zeigt *Abb. 5.3.1.* Achtung: Der Befehl „STIFTAB" ist nötig, wenn Sie zuletzt das Programm aus Abb. 5.2.11 ausgeführt haben, denn dort wird mit „SH" der Schreibstift gehoben. Dieser Zustand bleibt erhalten, auch wenn Sie den Gosi-Compiler neu aufrufen. Wenn man dagegen das Programm im Menue „2=starten" unter Gosi aufruft, wird die Schildkröte immer zurückgesetzt.

Wie wir bereits wissen, verwendet man Unterprogramme, um Programme übersichtlicher zu gestalten. Wenn in einem Programm öfters ein bestimmter Programmteil vorkommt, ist es auch sinnvoll, daraus ein Unterprogramm zu machen, man spart sich so viel Tipparbeit und natürlich auch Speicherplatz. In Gosi können beliebig viele solcher Prozeduren definiert werden, dadurch wächst nur die Symboltabelle an. Wenn genug Speicherplatz vorhanden ist, so ist dies kein Problem.

Nun kann man mit dieser Art einfacher Unterprogramme noch recht wenig anfangen, interessant wird es erst, wenn man auch Werte an das Unterprogramm übergeben kann.

5.4 Parameter

Als Parameterübergabe bezeichnet man den Vorgang, Werte in Variablen an Unterprogramme zu übergeben. Wir werden in diesem Abschnitt sehen, wie man dies in Gosi tun kann.

Es gibt in Gosi zwei Schreibweisen für Variablen. Normalerweise steht vor jedem Variablennamen ein Doppelpunkt. Nur wenn man einer Variablen mit dem Befehl „SETZE" einen Wert zuweist, so muß man dem Namen ein Anführungszeichen voranstellen. Am Doppelpunkt erkennt der Compiler, daß es sich um einen Variablennamen handelt. Er verwendet ihn zur Unterscheidung von Unterprogrammen bzw. Funktionen. Der Doppelpunkt bedeutet soviel wie „Inhalt von". Sehen wir uns dazu die folgenden zwei Programmzeilen an:

```
SETZE "ANTON 5
VORWAERTS :ANTON
```

Mit SETZE wird der Variablen ANTON der Wert 5 zugewiesen. In der nächsten Zeile wird die Variable verwendet. Der VORWAERTS-Befehl verlangt eine Schrittzahl. Die Anzahl der Schritte steht hier in der Variablen ANTON und beträgt 5.

Abb. 5.3.1
Mit dem LERNE-Befehl kann man neue Worte einführen

```
lerne quadrat
 wh 4 [vw 100 re 90]
ende

stiftab
wh 10 [quadrat li 36]
```

dyadische Operationen
Arithmetik

+	Addition
–	Subtraktion
!	Oder-Verknüpfung
*	Multiplikation
/	Division
\	Modulo-Operation
&	Und-Verknüpfung

Abb. 5.4.1
Dyadische Operationen

diverses

$	Sedezimal-Zahl
	Dezimal-Zahl
( )	
:	variable
"	text
[	text]
	funktion

Abb. 5.4.2
Diverse Operationen

dyadische Operationen
Vergleiche

=	gleich
<	kleiner
<=	kleiner gleich
>	größer
>=	größer gleich
<>	ungleich

Abb. 5.4.3
Vergleiche

```
lerne quadrat
 wh 4 [vw :groesse re 90]
ende

sa
setze "groesse 100
quadrat
setze "groesse 50
quadrat
```

Abb. 5.4.4
Parameterübergabe über Variable

LERNE name :var 1 :var 2 :var 3 ...

GOSI-Anweisungen

ENDE

Abb. 5.4.5 Die LERNE-Anweisung mit Parametern

```
lerne quadrat :groesse
 wh 4 [vw :groesse re 90]
ende

sa
quadrat 100
quadrat 50
```

Abb. 5.4.6 Die direkte Parameterübergabe

Neben Zahlen-Angaben, wie in unserem Beispiel, kann man in Gosi auch Arithmetik durchführen. Alle Variablen haben eine Wortbreite von 32 Bit, und normalerweise wird mit dieser Breite auch gerechnet.

Abb. 5.4.1 zeigt eine Zusammenstellung verschiedener dyadischer Operationen. Neben der Addition, Subtraktion, Multiplikation und Division finden sich auch logische Verknüpfungen, wie UND und ODER, die auch mit 32 Bit durchgeführt werden. Eine Modulo-Operation ist auch vorhanden; was man darunter versteht, haben wir bereits im Kapitel Pascal erfahren.

Das Plus- sowie das Minuszeichen kann außer den hier beschriebenen Operationen auch als Vorzeichen von Zahlen eingesetzt werden. Außerdem besteht die Möglichkeit, eine NICHT-Verknüpfung durchzuführen.

Abb. 5.4.2 zeigt weitere Möglichkeiten, mit Zahlen und Texten zu arbeiten. So werden sedezimale Zahlen von dezimalen durch das Voranstellen des Dollarzeichens unterschieden. Mit Klammern kann man die Reihenfolge von arithmetischen Operationen bestimmen. „:variable" kennen wir schon, Variable dürfen auch in Formeln auftauchen. Die Angaben "text und [text.] sind für die Textoperationen vorgesehen, die in einem gesonderten Abschnitt besprochen werden. Und dann kann man noch Funktionen aufrufen wie z. B. SIN und COS.

Abb. 5.4.3 zeigt der Vollständigkeit halber alle möglichen Vergleichs-Operationen, die für die Bildung von Bedingungen, aber auch für logische Ausdrücke verwendet werden können. Ist ein Vergleich wahr, so liefert er das Ergebnis $FFFFFFFF, wenn er falsch ist, so liefert er den Wert 0.

Prozeduren und Parameter

Mit SETZE könnte man nun Werte in eine Variable übertragen, um diese dann im Unterprogramm zu verwenden. Man würde dadurch also Parameter in das Unterprogramm übertragen. *Abb. 5.4.4* zeigt ein solches Beispiel. Diese einfache Methode birgt aber auch Gefahren in sich. So kann man bei großen Programmen schnell die Übersicht verlieren, wo welche Variable schon Verwendung fand. In manchen Programmiersprachen, so z. B. in Basic, ist dies die einzige Möglichkeit, Parameter zu übergeben.

In Gosi geht es eleganter. Bei der Procedure-Definition mit dem LERNE-Befehl kann man hinter dem Namen eine Liste von Parametern angeben. *Abb. 5.4.5* zeigt ein Schema. Dabei werden die gewünschten Variablennamen einfach hinter dem Namen aufgelistet. Im Prinzip können das beliebig viele sein. Diese Variablen kann man dann innerhalb der Prozedur frei verwenden. Beim Aufruf der Prozedur gibt man einfach, der Zahl der Variablen entsprechend, Werte mit an, und diese Werte werden an die Variablen übertragen. Die Variablen haben eine Besonderheit, man sagt: sie sind lokal. Das bedeutet, wenn der gleiche Name als Variable außerhalb der Prozedur verwendet wird und man diese Variable schon mit einem Wert belegt hatte, bleibt der Wert erhalten. Dazu werden alle lokalen Variablen vor dem Start der Prozedur auf den Stack gerettet.

Abb. 5.4.6 zeigt die abgeänderte Form des Programms aus Abb. 5.4.4. Jetzt findet die neue Parameterübergabe Anwendung. Sieht man sich das Assemblerlisting an, so ist zu

erkennen, daß gleich nach dem Beginn des Unterprogramms bei der Marke „QUADRAT" zuerst zwei Variablen auf den Stack gerettet werden. Die Variable „PRF−2" ist als Hilfsvariable für spezielle Gosi-Befehle (PRUEFE, WENNWAHR, WENNFALSCH) zuständig der Gosi-Compiler rettet diese Variable auch dann, wenn sie nicht benötigt wird.

Dann wird die Variable „GROESSE" auf den Stack gerettet. Bei der Marke „L2" wird der Inhalt des Registers D0 in diese Variable transportiert. Damit wird der Parameter übergeben. Am Schluß, hinter der Marke „L3", werden die alten Werte wieder restauriert. Beim Aufruf von QUADRAT legt Gosi den jeweiligen Parameter in das Register D0. Wenn weitere Parameter verwendet werden, so werden sie nacheinander in die Register D1,D2 ... bis D6 gelegt. Folgen dann noch mehr Parameter, so werden eigene Hilfsvariable erzeugt, die den Namen DX7, DX8 usw. tragen.

Die Anzahl der übertragenen Variablen wird nicht mit der Parameterliste verglichen, dadurch sind Fehler möglich. Man sollte daher die Anzahl immer genau kontrollieren. Auf die Prüfung wurde verzichtet, da man hier auch einfach Assembler-Unterprogramme einbinden kann. Der Gosi-Compiler übergibt die Werte einfach gemäß den Regeln.

Bevor wir uns an weitere komplexe Beispiele wagen, hier noch ein wichtiges Thema: Die Rekursion. Sie wird in Logo und somit auch in Gosi gerne verwendet. *Abb. 5.4.7* zeigt ein Beispiel. Ein Kreis soll definiert werden. Zunächst folgen die Anweisungen VORWAERTS 1 und RECHTS 1. Dann wird das Unterprogramm „KREIS" erneut aufgerufen. Da dies innerhalb der Prozedur „KREIS" geschieht, nennt man es Rekursion. Das Programm ruft sich selbst auf. Dieses Programm hört nicht mehr auf. Es fehlt das Abbruchkriterium. *Abb. 5.4.8* zeigt den erzeugten Assembler-Text. Vor der Marke „L3" erfolgt ein Sprung zurück zur Marke „L2". Der Teil zwischen „L2" und „L3" wird unbegrenzt ausgeführt. Zur Marke „L3" kommt das Programm nicht. Wir können das laufende Programm nur durch Drücken der Reset-Taste abbrechen.

Im Programm der *Abb. 5.4.9* ist ein Abbruchkriterium eingebaut. Dazu wurden drei neue Gosi-Befehle verwendet. „WENN" prüft eine hinter „WENN" angegebene Bedingung und führt dann, wenn diese Bedingung wahr ist, die in eckigen Klammern folgenden Anweisungen aus. In unserem Fall ist als Bedingung „TASTE?" angegeben. Hierbei handelt es sich um eine eingebaute Funktion, die den Wert WAHR liefert, wenn eine Taste gedrückt wurde. Als Anweisung für den Wahr-Fall geben wir „RUECKKEHR" an. Diese Anweisung bewirkt ein Verlassen des Unterprogramms. Startet man das Programm, so wird zunächst, wie vorher, der Kreis gezeichnet, und die Schildkröte hört nicht auf, sich entlang der Kreisbahn zu bewegen. Durch das Betätigen einer beliebigen Taste bleibt die Schildkröte stehen, und das Menue meldet sich wieder.

Abb. 5.4.10 zeigt den erzeugten Assembler-Text. Mit Hilfe der Anweisung „CSTS" wird die Tastenabfrage durchgeführt. Wird der Wert Null zurückgemeldet, so wird das Programm bei der Marke „L4" fortgesetzt, es wurde keine Taste gedrückt. War der Wert ungleich Null, so wird zur Marke „L3" gesprungen und dort die Rückkehr aus dem Unterprogramm eingeleitet.

Abb. 5.4.11 zeigt die allgemeine Form des „WENN"-Befehls. Wie zu erkennen ist, kann außer dem in unserem Beispiel angegebenen Ja-Teil auch optional ein Nein-Teil angegeben werden. Dieser ist ebenfalls in eckigen Klammern anzugeben und wird nur

Abb. 5.4.7
Die Rekursion in Gosi

```
GOSI68K Compiler Vers 3.0  (C) 1984 Rolf-Dieter Klein
*
*
*lerne kreis
* vw 1 re 1
* kreis
*ende
*
*sa                                          zu Abb. 5.4.8
*kreis
*
*
*--------------
UNTERPROGRAMM "KREIS
Rolf-D.Klein 68000/08 Assembler 4.3 (C) 1984, Seite 1


00931E                      ******************************
00931E                      * GOSI68K COMPILER VERS 3.0 *
00931E                      * (C) 1985, MUENCHEN        *
00931E                      * ROLF-DIETER KLEIN         *
00931E                      ******************************
```

```
00931E                          GOSI:
00931E  4EB9 000E3168            JSR @SETA5
009324  4EB9 000E3FEC            JSR @CLRSCREEN
00932A  4EB9 000E1342            JSR @FIRSTTIME
009330  6000 0028                BRA L1
009334                          KREIS:
009334  3F39 00009368            MOVE.W PRF_2,-(A7)
00933A                          L2:
00933A  7001                     MOVEQ.L #$1,D0
00933C  4EB9 000E1288            JSR @SCHREITE
009342  203C FFFFFFFF            MOVE.L #-$1,D0
009348  4EB9 000E12F6            JSR @DREHE
00934E  6000 FFEA                BRA L2
009352                          L3:
009352  33DF 00009368            MOVE.W (A7)+,PRF_2
009358  4E75                    RTS
00935A                          L1:
00935A  4EB9 000E1090            JSR @SENKE
009360  4EB9 00009334            JSR KREIS
009366  4E75                    RTS
009368  0000                    PRF_2: DC.W 0
00936A
00936A                          * ENDE GOSI-UEBERSETZUNG
00936A                           END

0E8B6E  Ende-Symboltabelle
```

Abb. 5.4.8 So werden Tail-Rekursionen übersetzt

```
lerne kreis
 vw 1 re 1
 kreis
ende

sa
kreis
```

Abb. 5.4.9 Das Abbruchkriterium in der Mitte

```
GOSI68K Compiler Vers 3.0  (C) 1984 Rolf-Dieter Klein
*
*
*lerne kreis
* wenn taste? [rueckkehr]
* vw 1 re 1
* kreis
*ende
*
*sa
*kreis
*
*
*--------------
UNTERPROGRAMM "KREIS
Rolf-D.Klein 68000/08 Assembler 4.3 (C) 1984, Seite 1


009372                  ******************************
009372                  * GOSI68K COMPILER VERS 3.0 *
009372                  * (C) 1985, MUENCHEN        *
009372                  * ROLF-DIETER KLEIN         *
009372                  ******************************
009372                  GOSI:
009372  4EB9 000E3168    JSR @SETA5
009378  4EB9 000E3FEC    JSR @CLRSCREEN
00937E  4EB9 000E1342    JSR @FIRSTTIME
009384  6000 003A        BRA L1
009388                  KREIS:
009388  3F39 000093CE    MOVE.W PRF_2,-(A7)
00938E                  L2:
00938E  4EB9 000E09D8    JSR @CSTS
009394  4880             EXT.W D0
009396  48C0             EXT.L D0
009398  6700 0006        BEQ L4
00939C  6000 001A        BRA L3
0093A0                  L4:
0093A0  7001             MOVEQ.L #$1,D0
0093A2  4EB9 000E1288    JSR @SCHREITE
0093A8  203C FFFFFFFF    MOVE.L #-$1,D0
0093AE  4EB9 000E12F6    JSR @DREHE
0093B4  6000 FFD8        BRA L2
```

Abb. 5.4.10 Das Ergebnis der Übersetzung

```
0093B8                    L3:
0093B8  33DF 000093CE      MOVE.W (A7)+,PRF_2
0093BE  4E75               RTS
0093C0                    L1:
0093C0  4EB9 000E1090      JSR @SENKE
0093C6  4EB9 00009388      JSR KREIS
0093CC  4E75               RTS
0093CE  0000              PRF_2: DC.W 0
0093D0
0093D0                    * ENDE GOSI-UEBERSETZUNG
0093D0                     END

0E8B6E  Ende-Symboltabelle
```

zu Abb. 5.4.10

```
WENN  bedingung  [ GOSI-Anweisungen ]
                    im Ja-Fall
```

Abb. 5.4.11
Der WENN-Befehl

```
WENN  bedingung  [ GOSI-Anweisungen im Ja-Fall ]
                 [ GOSI-Anweisungen im Nein-Fall ]
```

```
lerne poly :laenge :winkel
 wenn taste? [rueckkehr]
 vw :laenge re :winkel
 poly :laenge :winkel
ende

sä
poly 100 108
```

Abb. 5.4.12 Polygone

```
Textstart=009000 Fenster=009000 Tor=009000      amer CTRL-J=Hilfe
```

dann ausgeführt, wenn die Bedingung nicht erfüllt ist. Damit lassen sich elegant Fallunterscheidungen aufbauen. Die „WENN"-Anweisung läßt sich natürlich auch verschachteln, es kann also im Ja- oder Nein-Teil wieder eine „WENN"-Anweisung auftreten.

Um die unterschiedlichen Anwendungen der jetzt kennengelernten Anweisungen zu sehen, folgen nun ein paar Programmbeispiele aus der Schildkrötensprache.

Das Programm der *Abb. 5.4.12* kann unterschiedliche Polygone erzeugen. Probieren Sie auch mal andere Winkel (60,90,72,135,144) aus. Dabei muß der Wert „laenge" ggf. korrigiert werden. Versuchen Sie auch die Werte so zu dimensionieren, daß sich eine Kreisbahn ergibt.

Die *Abb. 5.4.13* zeigt ein weiteres Programm. Es zeichnet nach dem Start einen Stern auf den Bildschirm. Drückt man dann eine Taste, so erscheint zusätzlich ein Zahnrad, und bei nochmaligem Drücken das Menue. *Abb. 5.4.14* zeigt das gesamte Bild. Im Programm wird die Prozedur „NEUPOLY" zweimal aufgerufen. Das erste Mal mit den Parametern 50 und 144, also einer Länge von 50 und einem Winkel von 144, das zweite Mal mit 10 und 125. Damit beide Figuren nicht übereinander zu liegen kommen, wird mit AUFXY neu positioniert. Durch den Befehl AUFKURS bringen wir die Schildkröte in eine definierte Blickrichtung, da uns der aktuelle Winkel unbekannt ist. Dann folgt eine Anweisung:

SETZE "DUMMY TASTE

Damit wird eine Variable, hier DUMMY genannt, mit dem Wert der Funktion TASTE belegt. TASTE ist eine eingebaute Funktion, die den Wert des eingelesenen Zeichens liefert. Diese Anweisung ist nötig, da die Prozedur Neupoly in Abhängigkeit der Funktion TASTE? abgebrochen wird. Holt man nun nach dem Drücken einer Taste den Wert nicht ab, so liefert die Funktion TASTE? für den nächsten Aufruf schon von Anfang an den Wert wahr. Um zu verhindern, daß dem zweiten Aufruf von Neupoly gleich deren Abbruch folgt, laden wir den ASCII-Wert der betätigten Taste in eine Dummyvariable namens DUMMY.

Eine andere Möglichkeit, zu einem Abbruchkriterium zu kommen, zeigt *Abb. 5.4.15*. Man übergibt der Prozedur einfach die Anzahl der gewünschten Durchläufe in der Variablen MAXIMAL. Im Unterprogramm wird am Ende jedes Durchlaufs von diesem Wert eine Eins subtrahiert. Hat MAXIMAL den Wert Null erreicht, so wurde die vorgegebene Anzahl an Durchläufen ausgeführt, und ein Rücksprung kann eingeleitet werden. Das Vermindern von MAXIMAL geschieht beim rekursiven Aufruf von Polyspi. Hier werden auch die anderen Variabeln verändert, so daß sich spiralförmige Figuren (*Abb. 5.4.16*) ergeben.

Aufgaben:

1. Versuchen Sie, die verschiedenen möglichen Figuren zu klassifizieren, die sich bei unterschiedlichen Winkeln ergeben, also z. B. quadratförmig usw.!
2. Eine weitere Variante entsteht, wenn man nicht die Seitenlänge, sondern den Winkel erhöht. Ändern Sie das Programm entsprechend ab!

```
lerne neupoly :laenge :winkel
 wenn taste? [rueckkehr]
 vw :laenge re :winkel
 vw :laenge re (2 * :winkel)
 neupoly :laenge :winkel
ende

sh aufxy 100 100 sa
neupoly 50 144
sh aufxy 300 200 aufkurs 90 sa
setze "dummy taste
neupoly 10 125
```

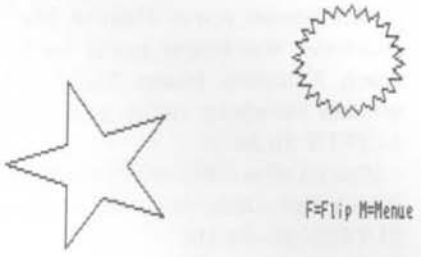


Abb. 5.4.14 Das Ergebnis auf dem Bildschirm

Abb. 5.4.13
Weitere Polygone

```
lerne polyspi :laenge :winkel :maximal
 wenn :maximal=0 [rueckkehr]
 vw :laenge re :winkel
 polyspi :laenge+3 :winkel :maximal-1
ende

sa
polyspi 1 121 70
polyspi 1 91 73
polyspi 1 181 100
```

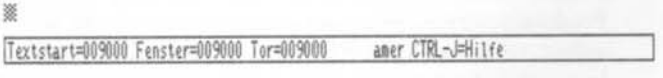


Abb. 5.4.15 Polyspiralen

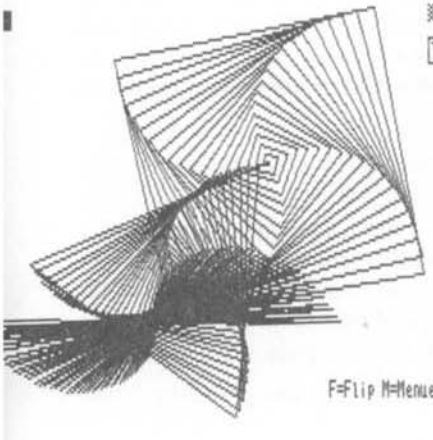


Abb. 5.4.16 Das Ergebnis auf dem Bildschirm

Jetzt wieder etwas Theorie. Die Übergabe von Parametern bleibt nicht ganz ohne Probleme. Wer Pascal kennt weiß, daß man dort Parameter in Klammern angibt und durch Kommata trennt. Das hat seinen besonderen Grund. Pascal erkennt daran, welcher Parameter wohin gehört. In Gosi ist das nicht so. Beispiel:
AUFRUF 10 20 30

Hier ist alles eindeutig; 10 gehört zum ersten Parameter, 20 zum zweiten und 30 zum dritten Parameter. Aber was ist beim
AUFRUF 10 -20 30?
Gosi rechnet wie folgt: 10 -20 gibt -10 also gilt:
AUFRUF -10 30,
und nun fehlt ein Parameter. Wenn man es so schreiben will, wie es oben gemeint ist, nämlich 10 als ersten und -20 als zweiten Parameter, so muß man in Gosi und natürlich auch in Logo schreiben:
AUFRUF 10 (-20) 30

Durch die Klammer wird es eindeutig. Also im Zweifelsfall lieber eine Klammer zuviel als eine zuwenig. Durch Klammern läßt sich immer eine eindeutige Schreibweise erreichen.

Abb. 5.4.17 zeigt ein Programmbeispiel. Hier wird die für uns noch neue, ebenfalls eingebaute Funktion „ZZ" verwendet. „ZZ" ist die Abkürzung für Zufallszahl. Diese Funktion benötigt als Parameter einen Wert, der das Maximum angibt. Die Zufallsfunktion entspricht dem Grundprogramm-Unterprogramm RND. Als Ergebnis wird ein Wert zwischen 0 und der angegebenen Zahl-1 geliefert. Hier sind die Klammern nötig, um dem Gosi-Compiler mitzuteilen, welche Parameter wohin gehören. *Abb. 5.4.18* zeigt einen möglichen Bildschirminhalt; das Programm läßt sich durch Drücken irgendeiner Taste beenden.

Bisher stand die Rekursion immer am Schluß einer Prozedur, in Logo und Gosi hat das einen besonderen Namen: „tail recursion". Diese Rekursion kann man nämlich durch einen Sprung ersetzen. Der Gosi-Compiler macht das automatisch. Das hat den Vorteil, daß sich der Computer nicht merken muß, wo er das Unterprogramm aufgerufen hat. Das spart Speicherplatz. Wenn man die Rekursion jedoch mitten in das Programm legt, so benötigt man einen Unterprogrammaufruf. Dabei wird, wie wir bereits wissen, der Stack benötigt; auf ihm werden alle Parameter und die aktuelle Programmadresse gesichert. Die Rekursionstiefe, also die Anzahl der Aufrufe von sich selbst, ist damit begrenzt. Also Vorsicht bei Rekursionen dieser Art! Das Programm kann sich leicht mal aufhängen, wenn man die Abbruchkriterien nicht sorgfältig auswählt.

Als kleine Einstimmung für rekursive Programme dient das Programm nach der *Abb. 5.4.19*, das *Abb. 5.4.20* erzeugt. Dieses Programm sieht auf den ersten Blick ganz harmlos aus, ist es aber nicht. Darin verborgen sind zwei nacheinander folgende, rekursive Aufrufe. Es wird zunächst ein Ziegel gezeichnet, und zwar der oberste. Die Rekursion besagt nun, daß das gleiche Gebilde um einen Ziegel nach unten versetzt, einmal auf die linke Hälfte und zum anderen auf die rechte Hälfte gezeichnet werden soll. Dadurch entsteht das turmartige Gebilde. Dabei werden Teile der Ziegelmauer auch mal doppelt gezeichnet.

```
lerne sternenhimmel
 wenn taste? [rueckkehr]
 sh aufxy (zz 512) (zz 512)
 setze "groesse (:zz 24)+6
 sa wh 6 [vw :groesse re 144]
 sternenhimmel
ende

sternenhimmel
```

Abb. 5.4.17 Zufallszahlen

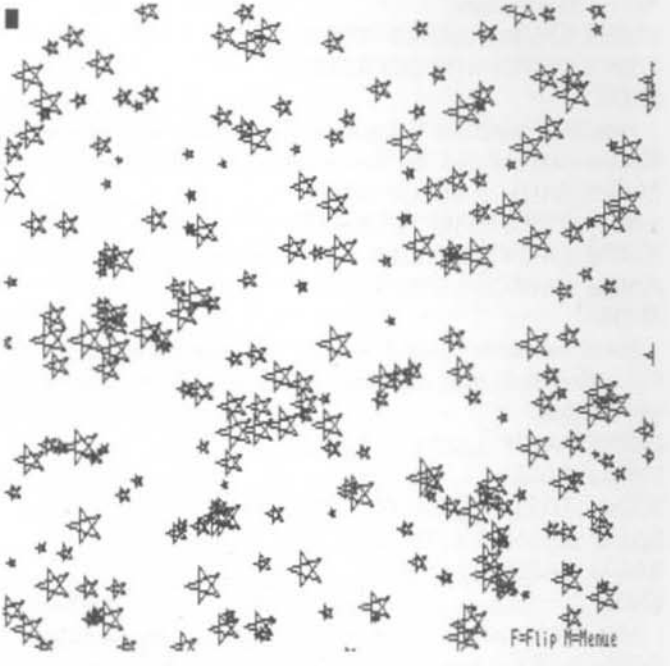


Abb. 5.4.18
Der Sternenhimmel

```
lerne ziegelturm :tiefe
 sa wh 4 [ vw 20 re 90 ]
 wenn :tiefe=0 [rueckkehr]
 sh rw 20 li 90 vw 10 re 90
 ziegelturm :tiefe-1
 sh re 90 vw 20 li 90
 ziegelturm :tiefe-1
 sh li 90 vw 10 re 90 vw 20
ende

ziegelturm 8
```

Abb. 5.4.19 Komplexe Rekursionen

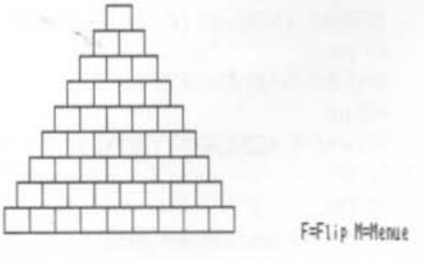


Abb. 5.4.20 Ergebnis von Abb. 5.4.19

Wie kommt man auf solche Programmlösungen? Leiten wir mal gemeinsam die Konstruktion eines rekursiv dargestellten Baums ab. *Abb. 5.4.21* zeigt unsere Wunschvorstellung. Der Baum soll einen Stamm besitzen, daran sind Zweige, die sich weiter verzweigen. Jeder Teilzweig soll kleiner werden, er soll sich auf die Hälfte verkürzen. Wie sieht das Programmgerüst aus?

```
LERNE BAUM :LAENGE :TIEFE
ENDE
```

Das ist das erste, grobe Schema. Der Baum soll zwei Parameter besitzen, einmal die Startlänge, also die Länge des Stammes, und dann die Rekursionstiefe, die über die Anzahl der Verzweigungen Auskunft gibt. Der nächste Schritt ist der Einbau eines Abbruchkriteriums.

```
LERNE BAUM :LAENGE :TIEFE
WENN :TIEFE=0 [RUECKKEHR]
ENDE
```

Nun fehlt noch der Rekursive Aufruf. Man wird hier zwei Aufrufe brauchen, denn am Stamm sitzt bei uns ein linker und ein rechter Zweig. Also lautet das Programm:

```
LERNE BAUM :LAENGE :TIEFE
WENN :TIEFE=0 [RUECKKEHR]
BAUM :LAENGE/2 :TIEFE-1
BAUM :LAENGE/2 :TIEFE-1
ENDE
```

Beim erneuten Aufruf soll ja nur die halbe Länge verwendet werden, und die Rekursionstiefe muß um eins verringert werden. Nun muß natürlich auch was gezeichnet werden:

```
LERNE BAUM :LAENGE :TIEFE
VW :LAENGE
WENN :TIEFE=0 [RUECKKEHR]
BAUM :LAENGE/2 :TIEFE-1
BAUM :LAENGE/2 :TIEFE-1
ENDE
```

Man würde jetzt eine gerade Linie als Ergebnis erhalten, denn vor dem neuen Baum-Aufruf muß ein neuer Winkel verwendet werden, wir wählen mal 30°, und nach dem Aufruf muß der Winkel wieder korrigiert werden.

```
LERNE BAUM :LAENGE :TIEFE
VW :LAENGE
WENN :TIEFE=0 [RUECKKEHR]
LI 30
BAUM :LAENGE/2 :TIEFE-1
RE 60
BAUM :LAENGE/2 :TIEFE-1
LI 30
ENDE
```

Starten wir einen ersten Versuch. *Abb. 5.4.22* zeigt das Programm. Doch schauen Sie sich *Abb. 5.4.23* an, wie ein Baum sieht das nicht aus, wir haben also einen Fehler gemacht. Klar, nachdem ein Zweig gezeichnet wurde, muß die Schildkröte wieder in

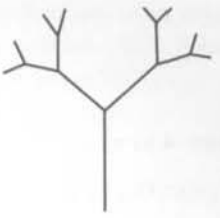


Abb. 5.4.21 Aufgabe: Dieser Baum soll gezeichnet werden

```
lerne baum :laenge :tiefe
 vw :laenge
 wenn :tiefe=0 [rueckkehr]
 li 30
 baum :laenge/2 :tiefe-1
 re 60
 baum :laenge/2 :tiefe-1
 li 30
 rw :laenge
ende

sa
baum 100 10
```

```
Textstart=009000 Fenster=009000 Tor=009000      amer CTRL-J=Hilfe
```

Abb. 5.4.22 Ein erster Versuch

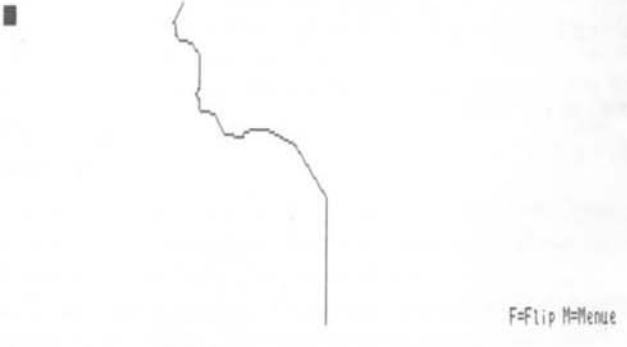


Abb. 5.4.23 Ein Programmfehler muß vorhanden sein

```
lerne baum :laenge :tiefe
 vw :laenge
 wenn :tiefe=0 [rueckkehr]
 li 30
 baum :laenge/2 :tiefe-1
 re 60
 baum :laenge/2 :tiefe-1
 li 30
ende

sa
baum 100 10
```

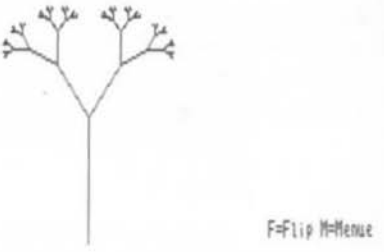


Abb. 5.4.25
Jetzt funktioniert das Programm

Textstart=009000 Fenster=009000 Tor=009000 aner CTRL-J=Hilfe

Abb. 5.4.24 Neuer Versuch mit dem Baumprogramm

```
lerne schachteldreiecke :laenge
 wenn :laenge<5 [rueckkehr]
 wiederhole 3 [
   schachteldreiecke :laenge/2
   vorwaerts :laenge rechts 120 ]
ende

schachteldreiecke 220
```

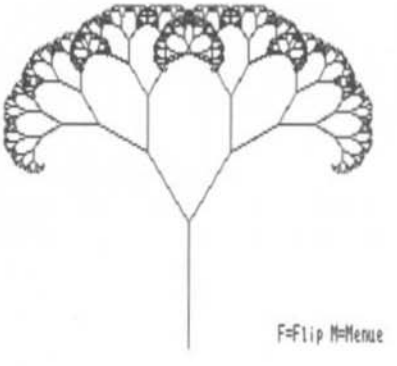


Abb. 5.4.26 Versuchen Sie diesen Baum zu zeichnen

Textstart=009000 Fenster=009000 Tor=009000 aner CTRL-J=Hilfe

Abb. 5.4.27 Eine weitere Rekursion

die Ausgangslage gebracht werden, also brauchen wir noch einen Befehl „RUECKWAERTS", der die Schildkröte wieder zurück bewegt. *Abb. 5.4.24* zeigt das geänderte Programm. In *Abb. 5.4.25* ist der fertige Baum dargestellt.

Aufgabe:

Versuchen Sie, einen Baum nach *Abb. 5.4.26* zu erzeugen!

Ein anderes rekursives Beispiel zeigt *Abb. 5.4.27*. Hier werden Dreiecke ineinander geschachtelt. Man erkennt im inneren Block die Struktur der Dreieckskonstruktion:
WIEDERHOLE 3 [VW :LAENGE RECHTS 120]

Außer dieser Konstruktion benötigen wir noch den rekursiven Aufruf und das Abbruchkriterium. *Abb. 5.4.28* zeigt das Ergebnis auf dem Bildschirm.

All diese Beispiele sollen Ihnen die Leistungsfähigkeit der rekursiven Programmiertechnik veranschaulichen, allerdings ist es nicht ganz einfach, solche Programme zu entwerfen, es ist schon etwas Programmiererfahrung erforderlich.

Rekursive Programme sind oft in der Lage, ganz erstaunliche Effekte zu produzieren. Am Rande sei erwähnt, daß der Gosi-Compiler selbst auch in rekursiver Programmiertechnik geschrieben ist.

Eine Anweisung besteht aus Anweisungen, oder ein arithmetischer Ausdruck kann aus mehreren arithmetischen Ausdrücken bestehen. All dies sind Formulierungen, die ganz schnell zu rekursiven Programmen führen.

Funktionen und Parameter

Eingebaute Funktionen haben Sie schon kurz kennengelernt, z. B. ZZ, TASTE oder TASTE?. Funktionen sind in der Lage, Werte zurückzuliefern, mit denen dann unmittelbar in einer Formel gerechnet werden kann.

Funktionen kann man aber auch selbst programmieren, dazu gibt es den Befehl „RUECKGABE", oder kurz „RG". Dahinter steht ein Parameter, nämlich der Wert, den die Funktion erhalten soll. Aus dem Pascal-Kapitel kennen Sie die Fakultät; wir wollen sie einmal in Gosi programmieren. Dazu benötigt man noch weitere Befehle in Gosi.

Die Eingabe einer Zahl kann man mit der Funktion „ZAHL" durchführen; sie verlangt als zusätzlichen Parameter die maximale Anzahl der einzulesenden Stellen. Dieser Stellenzahl entsprechend wird dann bei der Eingabeaufforderung ein Rahmen auf den Bildschirm gezeichnet.

Beispiel:
SETZE "A (ZAHL 20)

Auf dem Bildschirm erscheint ein 20 Zeichen breiter Rahmen. Man kann nun eine Zahl oder sogar eine Formel eingeben, und der Wert wird dann an die Variable A übergeben, nachdem man die Taste „CR" gedrückt hat.

Mit „DRUCKEZEILE" oder kurz „DZ" kann man Texte oder Variableninhalte auf dem Bildschirm ausgeben. Dazu erhält „DZ" eine Parameterliste, die beliebig viele Einträge beinhalten kann.

Beispiel:
DZ :A
druckt den Inhalt der Variablen „A" auf dem Bildschirm aus.
DZ [Hallo wie geht es]
druckt den Text „Hallo wie geht es" auf dem Bildschirm aus. Die eckigen Klammern darf man dabei nicht vergessen.
DZ [Die Variable A hat den Wert:] :A
druckt den Text „Die Variable A hat den Wert:" und den Inhalt der Variablen „A" in eine Zeile. „DZ" führt immer einen Zeilenvorschub aus, nachdem die Parameter ausgegeben sind. Will man das nicht haben, so verwendet man den Befehl „DRUCKE" oder kurz „DR", der sonst die gleiche Syntax besitzt.

Nun zurück zu unserem Fakultätsprogramm. *Abb. 5.4.29* zeigt das Programm. Es weist einige Besonderheiten auf. Das Eingabefeld verschwindet jeweils nach der Eingabe und wird stattdessen durch die Ausgabezeile überschrieben. Dies kommt daher, daß der Eingaberahmen die Cursorposition nicht verändert. Wurde also durch „DR" der Text „n eingeben:" auf den Bildschirm ausgegeben, so bleibt der Cursor direkt dahinter stehen. Der Eingaberahmen erscheint auch an dieser Stelle, jedoch bleibt er ohne Einfluß auf die Cursorposition. Der Rahmen wird durch das Eingabeprogramm automatisch gelöscht, und es bleibt nur der eingegebene Wert stehen. Dieser wird aber durch die danach folgende Ausgabe sofort überschrieben.

Experimentieren Sie ruhig einmal mit diesen Befehlen, und vergessen Sie nicht, die Klammern richtig zu setzen!

Sehen wir uns hier noch eine Reihe eingebauter Funktionen an.
ASC
liefert den Wert eines Zeichens im ASCII (DIN 66003). ASC "A liefert den Wert 65.
COS
liefert den gerundeten Wert des Cosinus mal 256; er entspricht dem Unterprogramm „COS", das im Grundprogramm eingebaut ist. Als Parameter wird ein Winkel in Grad angegeben. Entsprechend verhält es sich mit „SIN".
EINGABE
oder „EG" verlangt eine Zahl als Parameter und liefert die Adresse des Anfangs von dem Puffer, in dem die Zeichen stehen, doch zur Verwendung dieses Befehls kommen wir später. Die Funktion
KURS
benötigt keinen Parameter und liefert den aktuellen Winkel der Schildkröte in Grad.
MEM
liefert den Inhalt einer Speicherzelle, deren Adresse als Parameter angegeben wird. DZ (MEM $8000) druckt den Inhalt der Speicherzelle $8000 aus.
PORT
arbeitet ähnlich, nur daß hier IO-Ports angesprochen werden können: DZ (PORT $30) gibt den Inhalt des Ports mit der sedezimalen Adresse 30 aus.
TASTE
wartet so lange, bis eine Taste auf der Tastatur gedrückt wurde, dann liefert sie den ASCII des Zeichens, das der gedrückten Taste entspricht, ab. Mit
TASTE?

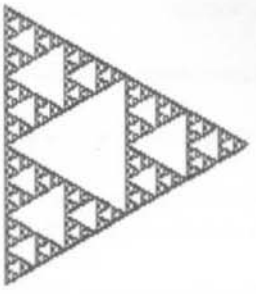


Abb. 5.4.28 Ein erstaunliches Ergebnis

```
lerne fakultaet :n
 wenn :n=0 [rueckgabe 1]
 rueckgabe (fakultaet :n-1)*:n
ende


lerne benutzer
 dr [n eingeben:]
 setze "n (zahl 20)
 wenn :n<0 [rueckkehr]
 dz [ Fak von ] :n [ ist ] (fakultaet :n)
 benutzer
ende

vi
benutzer
```

```
Textstart=009000 Fenster=009000 Tor=009000      aner CTRL-J=Hilfe
```

Abb. 5.4.29 Die Fakultät als Funktion

kann man prüfen, ob eine Taste gedrückt wurde. Die Funktion liefert den Wert WAHR, also $FFFFFFFF, falls eine Taste gedrückt wurde, und null, wenn dies nicht der Fall war. Wenn eine Taste gedrückt wurde, bleibt der Wert solange $FFFFFFFF, bis der Wert mit „TASTE" gelesen wurde.

XKO

liefert die aktuelle x-Koordinate der Schildkröte (Bereich 0..511) und

YKO

entsprechend die y-Koordinate (Bereich 0..511). Mit

ZAHL

kann man eine Zahl einlesen. Dazu wird als Parameter die Anzahl der Stellen eingegeben, die nachher als Rahmen ausgespart bleiben sollen. Und

ZZ

schließlich ist der Zufallsgenerator, der als Parameter die Zahl erhält, bis zu der Zufallszahlen geliefert werden sollen. Dabei wird die angegebene Zahl selbst nicht mehr verwendet. Der Wert liegt also im Bereich 0..Parameter-1.

Abschließend ein Programm, mit dem man dreidimensionale Funktionen graphisch darstellen kann. *Abb. 5.4.30* zeigt das Listing. Die erste LERNE-Anweisung definiert dabei die Funktion „FUN", die in Abhängigkeit von x und y einen Funktionswert liefert. Der Rest des Programms dient der graphischen Ausgabe. *Abb. 5.4.31* zeigt das erzeugte Bild, das übrigens, dank der schnellen Arithmetik von Gosi, innerhalb weniger Sekunden auf dem Bildschirm erzeugt wird.

Um sich weiter mit diesem Thema zu befassen, ist es ratsam, mit eigenen Funktionen zu experimentieren. Zu beachten ist natürlich, daß sie in den Wertebereich passen.

Hier noch einen Hinweis zum ersten Befehl, dem „OPTION"-Befehl, im Programm der Abb. 5.4.30. Wir hatten ihn bisher noch nicht verwendet. In Klammern können verschiedene Optionen stehen. Die Option „M" bedeutet, daß Multiplikation und Division nicht wie sonst üblich mit 32 Bit durchgeführt werden, sondern mit 16 Bit. Es werden dann direkt die Befehle MULS und DIVS des 68000/68008 verwendet. Dies bedeutet einen erheblichen Geschwindigkeitsgewinn. Läßt man diesen Befehl weg, so dauert die Darstellung einige Sekunden länger.

Mit der Verwendung dieser Option schränkt man natürlich den Zahlenbereich der Arithmetik ein. Es ist also darauf zu achten, daß man nicht durch zu große Zahlen einen Überlauf produziert.

Weitere Optionen und Befehle von Gosi finden Sie im Anhang. Wir wollen uns nun noch mit ein paar Spezialitäten von Gosi beschäftigen, die in Logo nicht in dieser Art vorkommen.

5.5 Felder

In Logo kennt man nur sogenannte Listenstrukturen. Die Listenstruktur ist ungeheuer leistungsfähig, aber auch schwer optimal zu realisieren, daher wurde in Gosi nicht die Listenstruktur, sondern die Feldstruktur als Datenstruktur gewählt. Ein Feld ist eine lineare Anordnung einzelner Variablen, die über einen gemeinsamen Namen in Verbin-

```
option [M]

lerne fun :x :y
 rg (sin(:x)+sin(:y))/20
ende
vi sa
setze "xa 0 setze "ya 0
setze "va 2

lerne inner :y
 wenn :y > :y2 [rk]
 setze "res (fun :x :y)
 setze "yy :res+(:y-:y1)/12+200-:h0
 setze "nh :nh+1
 setze "xx :h0+:nh*5
 wenn :first=1 [linie :xa :ya :xx :yy    ]
 setze "xa :xx setze "ya :yy
 setze "first  1
 inner :y+10
ende

lerne outer :x
 setze "first 0
 wenn :x > :x2 [rk]
 setze "h0 (:x-:x1)/5
 setze "nh -1
 inner :y1
 outer :x+10
ende

setze "x1 -360
setze "x2 400
setze "y1 -360
setze "y2 400

outer :x1

lerne inner2 :y
 setze "first 0
 wenn :y > :y2 [rk]
 setze "h0 0
 setze "nh :nh+1
 outer2 :x1
 inner2 :y+10
ende
```

Abb. 5.4.30
Programm zur graphischen Darstellung von Funktionen

```
lerne outer2 :x
 wenn :x > :x2 [rk]
 setze "res (fun :x :y)
 setze "yy :res/2+(:y-:y1)/12+200-:h0
 setze "xx :h0+:nh*5
 setze "h0 (:x-:x1)/5
 wenn :first=1 [linie :xa :ya :xx :yy    ]
 setze "xa :xx setze "ya :yy
 setze "first 1
 outer2 :x+10
ende


setze "nh -1
setze "x1 -360
setze "x2 400
setze "y1 -360
setze "y2 400

inner2 :y1

ende
```

zu Abb. 5.4.30

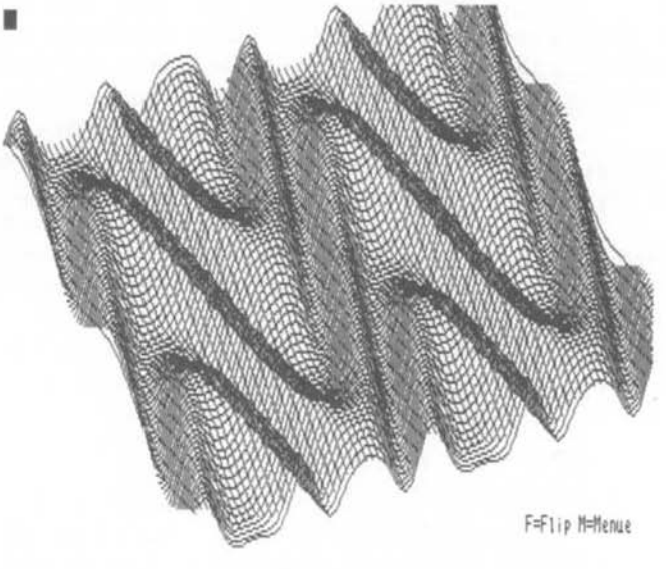


Abb. 5.4.31 Dieses Bild entsteht innerhalb von Sekunden

dung mit einem Index erreicht werden. Der Index gibt durch seinen Wert an, welche Variable gemeint ist.

Abb. 5.5.1 zeigt ein Schema. Die Anzahl der Feldelemente muß allerdings vor Benutzung des Feldes festgelegt werden. Dazu gibt es in Gosi den neuen Befehl „FELD". Dahinter wird der Feldname, gefolgt von der Feldgröße, angegeben. Die Feldgröße muß eine Konstante, keine Variable sein. Jedes Feldelement kann 32 Bit aufnehmen. Beispiel:

FELD "ALPHA 100

definiert ein Feld mit 101 Feldelementen, dß das erste Feldelemente, den Index 0 trägt und so von 0 bis 100, 101 Feldelemente angesprochen werden können.

Auf Feldelemente kann man durch die Angabe eines Index zugreifen. Dieser Index wird in eckigen Klammern direkt hinter den Namen gesetzt. Sonst werden die Felder genauso behandelt wie Variable. Felder kann man nicht als lokale Variable in Unterprogrammen verwenden. Sie müssen im Hauptprogramm deklariert werden und dürfen nicht in der LERNE-Zeile auftauchen. Man kann sie natürlich als Werte an Unterprogramme übergeben, wenn man sie in der Aufrufliste angibt, auch hier nur mit einer Indexangabe. Der Index kann übrigens auch berechnet werden, also selbst Variable oder Felder und Formeln enthalten.

Die *Abb. 5.5.2* zeigt ein einfaches Beispiel. Das Feld „ALPHA" besitzt 101 Feldelemente. Die erste Prozedur belegt das gesamte Feld mit dem Wert 0, denn die Feldelemente sind nach dem Start von Gosi zunächst undefiniert. Die Prozedur „DRUCKEFELD" gibt alle Feldelemente aus. Es müßten also 100 Nullen auf dem Bildschirm untereinander erscheinen.

Mit solchen Feldern kann man interessante Dinge unternehmen. Stellen wir uns eine Aufgabe aus dem Bereich der Statistik. Wir wollen die Häufigkeit der Summen zweier Zufallszahlen in einer Balkengraphik darstellen. Vereinfacht gesagt möchten wir wissen, wie groß die Wahrscheinlichkeit beim Arbeiten mit zwei Würfeln ist, daß die Summe der Würfe 2,3,4,5...12 ergibt. Wir arbeiten allerdings nicht mit richtigen Würfeln, die ja nur eine Zufallszahl zwischen 1 und 6 liefern, sondern mit unserem Zufallsgenerator, von dem wir uns Zahlen zwischen 0 und 125 liefern lassen. Die Summen können also zwischen 0 und 250 betragen. Für jede mögliche Summe sehen wir ein Element in unserem Feld vor. Immer wenn diese Summe auftritt, werden wir den entsprechenden Feldinhalt um eins erhöhen. Wir richten also für jede Möglichkeit einen Zähler ein. Unser Feld ist mit 251 Elementen (0 bis 250) zu definieren. Wenn also die Summe 10 auftaucht, so wird der Inhalt des Feldelementes mit dem Index 10 um eins erhöht. Die *Abb. 5.5.3* zeigt das Programm.

Der aktuelle Stand der Felder wird auf dem Bildschirm dargestellt. Diese Aufgabe läßt sich nur mit Feldern sinnvoll lösen. Das Feld wird vor Benutzung noch gelöscht und dann geht's los. *Abb. 5.5.4* zeigt ein mögliches Ergebnis.

Abb. 5.5.1
Organisation von Feldern

```
feld "alpha 100


lerne loeschefeld :startwert
 setze "alpha[:startwert] 0
 wenn :startwert=0 [rk]
 loeschefeld :startwert-1
ende


lerne druckefeld :startwert
 dz :alpha[:startwert]
 wenn :startwert=0 [rk]
 druckefeld :startwert-1
ende

loeschefeld 100
druckefeld 100
```

Textstart=009000 Fenster=009000 Tor=009000 amer CTRL-J=Hilfe

Abb. 5.5.2 Befehle für die Verarbeitung von Feldern

```
feld "ergebnis 250

lerne loeschefeld :n
 setze "ergebnis[:n] 0
 wenn :n=0 [rk]
 loeschefeld :n-1
ende

lerne statistik :zaehler
 wenn :zaehler=0 [rk]
 setze "a (zz 125)+(zz 125)
 setze "ergebnis[:a] :ergebnis[:a]+1
 linie :a*2 0 :a*2 :ergebnis[:a]
 statistik :zaehler-1
ende

loeschefeld 250
statistik 10000
```

Abb. 5.5.3
Beispiel mit graphischer Ausgabe

Textstart=009000 Fenster=009000 Tor=009000 amer CTRL-J=Hilfe

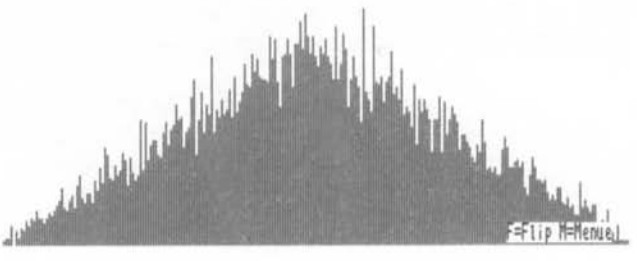


Abb. 5.5.4
Bildschirmausgabe

5.6 Zeichenverarbeitung

Textverarbeitung ist ein wichtiges Anwendungsgebiet für Mikrocomputer. Daher gibt es hierfür in Gosi entsprechende Befehle.

Zunächst kann man Zeichenfelder definieren. Dies geschieht mit dem Befehl „ZFELD", der genauso arbeitet wie „FELD", nur daß hier 1Byte-Feldelemente reserviert werden. Die „SETZE"-Anweisung hat für Zeichenfelder auch neue Eigenschaften. So kann man ganze Zeichenfelder kopieren. Beispiel:

```
ZFELD "NAME1 80
ZFELD "NAME2 80
...
SETZE "NAME1 :NAME2
```

Der Inhalt des Zeichenfeldes „NAME2" wird in das Zeichenfeld „NAME1" kopiert. Es wird dabei Elementinhalt für Elementinhalt übertragen. In den Zeichenfeldern muß ein Text stehen. Jeder Text muß mit einer 0 abgeschlossen werden. Die 0 bestimmt das Ende des Kopierens. Einem Zeichenfeld kann man auch einen festen Text zuweisen. Beispiel:

```
SETZE "NAME1 [Hallo wie geht es]
SETZE "NAME1 "Meier
```

Dabei kann der Text entweder in eckigen Klammern stehen und beliebige Zeichen, auch Leerzeichen enthalten, oder er wird mit einem Anführungzeichen geschrieben und besteht dann aus Zeichen, die kein Leerzeichen enthalten. Es wird automatisch je ein Zeichen in ein Feldelement abgelegt. Eine weitere Form ist die Zuweisung, beginnend bei einem Feldelement:

```
SETZE "NAME1 5 "Hallo
```

Ab Element 5 wird der Text „Hallo" abgelegt. Man kann auch Elemente kopieren. Beispiel:

```
SETZE "NAME1 5 :NAME1 3
```

Der Inhalt des Elements 3 wird in das Element 5 kopiert.

Jetzt zum Ausdruck von Zeichenfeldern. Dies kann ebenfalls durch „DZ" oder „DR" geschehen, dabei wird der ganze Text ausgegeben. Der Feldname muß dabei, ohne in eckige Klammern eingeschlossen zu sein, angegeben werden. Gibt man einen Index an, so wird der ASCII-Wert des Elementinhalts ausgegeben. Will man ein Einzelzeichen als Zeichen ausgeben, so muß man den Befehl „ZEICHEN" verwenden. Beispiel:

```
ZEICHEN :NAME1 5
```

gibt das Element 5 als Zeichen auf den Bildschirm.

Will man ein Feld ab einer bestimmten Adresse ausgeben, so geht dies auch, man muß allerdings zu einem Trick greifen. Die Angabe :NAME1 liefert die Adresse des Zeichenfeldes als Ergebnis. Mit dieser Adresse können wir dann weiterrechnen.

```
DZ :NAME1+3
```

druckt damit das Zeichenfeld „NAME1" beginnend mit dem Element 3 aus.

Abb. 5.6.1 zeigt ein Programmbeispiel. Ein Text soll im Sperrdruck ausgegeben werden. Dazu wird ein Unterprogramm „SPERRDRUCK" definiert, das den Inhalt des Zeichenfeldes mit dem Namen „HILFSFELD" im Sperrdruck ausgibt. HILFSFELD ist global definiert, in Gosi gibt es keine lokalen Zeichenfelder. Der Text in diesem

```
▒
▒


zfeld "hilfsfeld 80


lerne sperrdruck :index
 wenn :hilfsfeld[:index]=0 [rk]
 zeichen :hilfsfeld[:index] dr [ ]
 sperrdruck :index+1
ende

setze "hilfsfeld [Hallo Sperrdruck]
sperrdruck 0
dz
setze "hilfsfeld (eingabe 30)
dz
sperrdruck 0


▒
▒
Textstart=009000 Fenster=009000 Tor=009000      aner CTRL-J=Hilfe
```

Abb. 5.6.1 Beispiel für die Textverarbeitung

```
H a l l o   S p e r r d r u c k

E i n g a b e   e r f o l g t

                                                F=Flip M=Menue
```

Abb. 5.6.2 Ein Bildschirmbeispiel nach der Eingabe

```
zfeld "alles 80*101
zfeld "hilf 80

lerne loeschefeld :n
 setze "alles[:n*80] 0
 wenn :n=0 [rk]
 loeschefeld :n-1
ende

lerne einlesefeld :n
 setze "alles[:n*80] (eingabe 30)
 wenn :alles[:n*80]=0 [rk]
 wenn :n=0 [rk]
 einlesefeld :n-1
ende

lerne sortierefeld :n :merker
 wenn :alles[:n*80]<>0 [
  wenn :alles[:n*80]>:alles[(:n-1)*80]
   [ setze "hilf :alles+:n*80
     setze "alles[:n*80] :alles+(:n-1)*80
     setze "alles[(:n-1)*80] :hilf
     setze "merker 1 ]
 ]
  wenn :n=1 [ wenn :merker=0 [rk]
              [ setze "merker 0
                setze "n 101 ]]
  sortierefeld :n-1 :merker
ende

lerne ausgabefeld :n
 wenn :alles[:n*80]<>0 [dz :alles+:n*80 ]
 wenn :n=0 [rk]
 ausgabefeld :n-1
ende

vi
loeschefeld 100
einlesefeld 100
ausgabefeld 100
dz [bitte warten...]
sortierefeld 100 0
dz [Ergebnis: ]
ausgabefeld 100
```

Abb. 5.6.3
Texte sortieren mit dem Bubble-Sort

Hilfsfeld besteht aus zwei Teilen, dem fest vorgegebenen Teil „Hallo Sperrdruck", sowie einem von der Tastatur eingelesenen.

Das Programm gibt nach dem Start den ersten Text aus und erwartet dann eine Eingabe. In *Abb. 5.6.2* sieht man den Bildschirm nach erfolgter Eingabe des Textes „Eingabe erfolgt".

Jetzt werden wir uns einem komplexeren Beispiel zuwenden. Namen sollen ein- und anschließend in alphabetischer Reihenfolge wieder ausgegeben werden. Es muß also ein Sortiervorgang erfolgen Die Namen werden in einem Zeichenfeld gespeichert, das mehrere Texte aufnehmen kann. Dann kommt der sogenannte Bubble-Sort zum Einsatz. Dieses Verfahren ist einfach zu verstehen, aber sehr langsam. Ein wesentlicher Vorteil von Gosi für unser Vorhaben ist die Möglichkeit, Texte direkt zu vergleichen.

Abb. 5.6.3 zeigt das Programm. Es werden zwei Zeichenfelder definiert, eines mit dem Namen „ALLES", und das zweite „HILF". Das Unterprogramm „LOESCHEFELD" belegt jedes achzigste Element von „ALLES" mit einer 0. Da wir nur über eindimensionale Felder verfügen, muß man dieses Feld in Bereiche unterteilen. Durch eine Multiplikation des Index mit 80 kann man dies erreichen. Somit verhält sich das Ganze wie ein zweidimensionales Feld.

Im Unterprogramm „EINLESEFELD" werden die vom Benutzer eingegebenen Texte im „ALLES"-Feld abgelegt. Die Eingabe ist beendet, wenn entweder eine leere Eingabe (nur „CR") durchgeführt wurde, oder alle Feldelemente belegt sind. Dies ist dann der Fall, wenn „N" gleich null ist. Das Feld wird von hinten nach vorne gefüllt, also mit N=100 angefangen. Die Felddefinition wurde mit 101·80 durchgeführt. Es wird für N=100 an der Stelle 100·80 begonnen, das Feld zu beschreiben. Dieser Eintrag könnte bei einer Definition 100·80 über das Feld hinausragen.

Das Unterprogramm „SORTIERFELD" hat die Aufgabe, die eigentliche Sortierung durchzuführen. Dabei werden paarweise Feldelemente so lange vertauscht, bis die Reihenfolge des gesamten Feldes alphabetisch ist.

Im Unterprogramm „AUSGABEFELD" werden die Zeichenketten dann ausgegeben. Vor der Ausgabe wird dabei geprüft, ob überhaupt ein Eintrag vorhanden ist.

Das Hauptprogramm ruft hintereinander die Unterprogramme LOESCHEFELD, EINLESEFELD und dann AUSGABEFELD auf. Vor dem Aufruf von SORTIEREFELD wird dann der Text „bitte warten..." auf dem Bildschirm angezeigt, denn das Sortierverfahren ist sehr langsam. Schließlich wird nach dem Text „Ergebnis:" das sortierte Feld ausgegeben.

Aufgaben:

1. Eine erste Verbesserung der Geschwindigkeit kann man vornehmen, indem man nicht die Zeichenketten selbst vertauscht, sondern die Vertauschung in einem eigenen Indexfeld durchführt. Dies ist dann ein normales Feld, indem am Anfang die Werte 0,1,2,3,4,5,...100 stehen. Diese Elemente werden dann jeweils vertauscht, und jeder Zugriff auf das Zeichenfeld erfolgt, indem man zunächst den Wert aus dem Indexfeld holt und dann erst die Umrechung auf das Zeichenfeld durchführt, also :ALLES (:INDEX :N) 0.

Schreiben Sie das Programm dazu!
2. In der Literatur sind verschiedene Sortierverfahren beschrieben; versuchen Sie den „QUICKSORT" zu implementieren, z. B. aus (Wirth, Systematisches Programmieren, siehe Anhang).

5.7 Listenstrukturen

Sie haben nun einiges über Felder gehört. Noch mehr als diese Technik ist in Logo die Listenstruktur gebräuchlich. Was eine Listenstruktur ist, und wie man damit umgeht, erfahren Sie in diesem Abschnitt. Dabei ist die Listenstruktur in Gosi nicht direkt implementiert, aber Sie können sie mit Hilfe einiger Unterprogramme nachbilden, um so Erfahrungen damit zu sammeln. Wir fangen mit unserem bereits bekannten Sortierbeispiel an.

Abb. 5.7.1 zeigt den Aufbau eines Listenelementes, so wie wir es dazu brauchen. Da gibt es einen Zeiger mit dem Namen „Basis", er zeigt auf das erste Element. „Zeigt auf" bedeutet einfach, daß der Inhalt der Variablen „Basis" die Adresse des Anfangs des ersten Listenelementes ist.

Das Listenelement soll hier drei Einträge besitzen, zum einen zwei Zeiger und dann ein kleines Feld mit dem Text, der sortiert werden soll. Die beiden Zeiger haben nun eine besondere Bedeutung. Der linke Zeiger „L" soll auf das nächste Element zeigen, das alphabetisch „kleiner" ist als der Eintrag im Listenelement, und der Zeiger „L" soll auf das nächste Element zeigen, das alphabetisch „größer" ist als der Eintrag.

Wie kann man so etwas realisieren? Am einfachsten, man macht daraus Felder, und die Zeiger werden zu einem Index. Man benötigt ferner einen Zeiger, der auf nichts zeigt, um zu kennzeichnen, daß kein weiterer Eintrag folgt. Diesen Zeiger nennt man dann auch „NIL".

Dann können wir das Ganze in Gosi realisieren. Da Textfeld und Indexfeld verschiedene Datengrößen (8 Bit und 32 Bit) enthalten, muß man sie in getrennten Feldern speichern und benötigt noch einen zusätzlichen Zeiger, der die beiden verknüpft. *Abb. 5.7.2* zeigt ein Beispiel mit ein paar Einträgen. Dabei hängt die Anordnung der Zeiger von der Reihenfolge der Texteingabe ab.

Zu beachten ist, daß für das Programm mehr als 16KByte, mindestens 24KByte benötigt werden.

Abb. 5.7.3 zeigt das Programm und *Abb. 5.7.4* ein Eingabebeispiel. Der eigentliche Sortiervorgang findet hier schon während der Eingabe statt, daher ist die Routine „EINTRAGFELD" auch so groß geworden. Das Unterprogramm „SORTAUS" gibt das Feld in alphabethischer Reihenfolge aus.

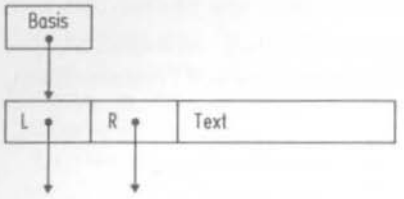


Abb. 5.7.1
Aufbau eines Listenelements

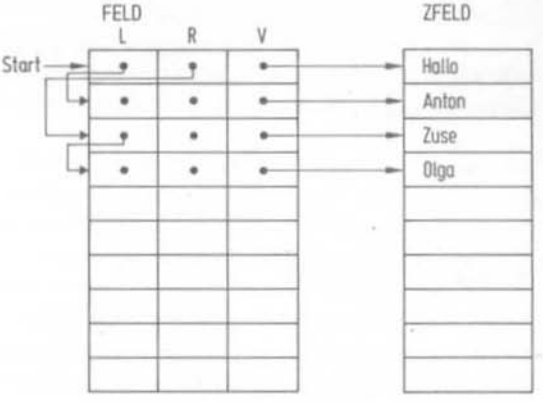


Abb. 5.7.2 Beispiel mit Einträgen

```
lerne ausgabefeld :n
 wenn :alles[:n*80]<>0 [dz :alles+:n*80 ]
 wenn :n=0 [rk]
 ausgabefeld :n-1
ende

lerne sortaus :index
 wenn :pfeld[:index*3]<>999 [
  sortaus :pfeld[:index*3]  ]
 wenn :pfeld[:index*3+2]<>999 [
  dz :alles+:pfeld[:index*3+2]*80 ]
 wenn :pfeld[:index*3+1]<>999 [
  sortaus :pfeld[:index*3+1]  ]
ende


vi
setze "frei 0
loeschefeld 100
einlesefeld 100
ausgabefeld 100
dz [Ergebnis: ]
sortaus 0

feld "pfeld 3*100
zfeld "alles 80*101
zfeld "hilf 80
```

zu Abb. 5.7.3

```
lerne loeschefeld :n
 setze "pfeld[:n*3] 999
 setze "pfeld[:n*3+1] 999
 setze "pfeld[:n*3+2] 999
 setze "alles[:n*80] 0
 wenn :n=0 [rk]
 loeschefeld :n-1
ende

lerne eintragfeld :index :n
 wenn :frei=0 [
  setze "pfeld[:index*3+2] :n
  setze "frei :frei+1
  rueckkehr
 ]
 wenn (:alles+:n*80)<(:alles+:pfeld[:index*3+2]*80)
  [ wenn :pfeld[:index*3]=999
     [
       setze "pfeld[:index*3] :frei
       setze "pfeld[:frei*3+2] :n
       setze "frei :frei+1
       rueckkehr
     ]
     [
        setze "index :pfeld[:index*3]
     ]
   ]
   [ wenn :pfeld[:index*3+1]=999
     [
        setze "pfeld[:index*3+1] :frei
        setze "pfeld[:frei*3+2] :n
        setze "frei :frei+1
        rueckkehr
     ]
     [
        setze "index :pfeld[:index*3+1]
     ]
   ]
 eintragfeld :index :n
ende
```

```
hallo
test
mit
listen
Struktur
Anton
0 Zahl
1 Zahl
Zuse
zwei
Ergebnis:

0 Zahl
1 Zahl
Anton
Struktur
Zuse
hallo
listen
mit
test
zwei

F=Flip M=Menue
```

Abb. 5.7.4
Ein Beispiel
auf dem Bildschirm

```
lerne einlesefeld :n
 setze "alles[:n*80] (eingabe 30)
 eintragfeld 0 :n
 wenn :alles[:n*80]=0 [rk]
 wenn :n=0 [rk]
 einlesefeld :n-1
ende
```

Abb. 5.7.3 Das fertige Sortierprogramm mit Listenverarbeitung

Aufgabe:

Analysieren Sie das Programm, und stellen Sie Struktogramme dafür auf!

5.8 Echtzeitaufgaben

Interessant ist Gosi auch bei Echtzeitaufgaben, wie bei der Erfassung von Meßwerten, oder beim Steuern und Regeln.

Für solche Aufgaben gibt es in Gosi die Befehle PORT und MEM jeweils in zwei Formen:

PORT adresse wert
MEM adresse wert
SETZE "variable PORT adresse
SETZE "variable MEM adresse

In den ersten beiden Fällen finden PORT und MEM als Befehl Verwendung. Die angegebene Adresse wird mit „wert" belegt. Bei der SETZE-Anweisung werden sie als Funktion eingesetzt. Der Inhalt des Ports oder Speichers wird eingelesen. Dabei wird immer ein Byte bearbeitet. Dies ist für Steuerungsaufgaben eine gute Einrichtung.

Abb. 5.8.1 zeigt ein einfaches Programm. Aufgabe des Programms ist es, den Inhalt des Ports mit der Adresse $30 auf den Bildschirm zu geben, und zwar in binärer Schreibweise. Dazu gibt es zunächst ein Unterprogramm „BINAERAUS", das einen Wert n in binärer Schreibweise ausgibt. Dieses Programm ist rekursiv programmiert und arbeitet nach dem Prinzip der fortgesetzten Division durch 2. Dazu ein Beispiel. Die Zahl 44 soll in das Zweiersystem umgewandelt werden. Dazu verfährt man wie folgt:

44 / 2 gibt 22 Rest 0
22 / 2 gibt 11 Rest 0
11 / 2 gibt 5 Rest 1
5 / 2 gibt 2 Rest 1
2 / 2 gibt 1 Rest 0
1 / 2 gibt 0 Rest 1

Die Zahl lautet dann: 101100. Die zuerst ermittelte Stelle ist die letzte Stelle des Ergebnisses. Im Programm wird die Restbildung mit der Modulo-Funktion durchgeführt, die durch den umgekehrten Divisionsstrich dargestellt ist. Wenn die Zahl N modulo 2 0 ist, also bei der Division einer Zahl N mit 2 kein Rest bleibt, so ist die betreffende Stelle der Dual-Zahl 0, sonst 1, dabei wird aber zunächst der Algorithmus für die Zahl N/2 (ganzer Teil) durchgeführt, um die Reihenfolge der Ausgabe von der höherwertigen zu niederwertigen Stelle zu garantieren.

Das Unterprogramm „AUSGABE" ruft die „BINAERAUS"-Routine auf, und zwar mit dem Inhalt des Ports $30 als Parameter. Wenn man dort die IOE-Baugruppe installiert und ein Tastenfeld anschließt, kann man die Lage der einzelnen Bits ansehen. Eine gedrückte Taste liefert dann den Wert 0. Achtung, die höherwertigen Stellen, die 0 sind, werden nicht ausgegeben. Daher erscheint der Wert 11110 auf dem Schirm, der einer Portbelegung von 00011110 entspricht.

```
lerne binaeraus :n
 wenn :n=0 [rk]
 wenn :n\2=0
  [ binaeraus :n/2 dr [0]]
  [ binaeraus :n/2 dr [1]]
ende


lerne ausgabe
 wenn taste? [rk]
 blinker 0 0
 binaeraus (port $30) dz [       ]
 ausgabe
ende

ausgabe
```

```
Textstart=009000 Fenster=009000 Tor=009000      amer CTRL-J=Hilfe
```

Abb. 5.8.1 Ausgabe eines IO-Portes in binärer Schreibweise

```
lerne testeio :n
 wenn taste? [rk]
 port $30 :n
 testeio (:n+1)&$ff
ende


testeio 0
```

```
Textstart=009000 Fenster=009000 Tor=009000      amer CTRL-J=Hilfe
```

Abb. 5.8.2 Ausgabe auf den Port $FFFFFF30

```
lerne holead
 @getad10
ende

lerne anzeige :pos
 wenn taste? [rk]
 sh mitte aufkurs :pos
 sa
 stift 1
 vw 100
 setze "pos (holead)
 sh mitte aufkurs :pos
 sa
 stift 0
 vw 100
 anzeige :pos
ende

vi
anzeige 90

```

```
Textstart=009000 Fenster=009000 Tor=009000      amer CTRL-J=Hilfe
```

Abb. 5.8.3 Programm mit A/D-Umsetzer-Daten

```


option [M]

lerne ellipse :n
 @setda  0 (cos :n)/3+128 (sin :n)/3+128
 ellipse (:n+1)\360
ende


ellipse 0



```

```
Textstart=009000 Fenster=009000 Tor=009000      amer CTRL-J=Hilfe
```

Abb. 5.8.5 Ansteuerung von D/A-Umsetzern

```
lerne zahnrad
 wh 24  [
 vw 20
 re 125
 vw 20
 re 250
 ]
ende


vi
seite 0 0
zahnrad
sh mitte
re 55 schr16tel 74 li 55
aufkurs 90-4
sa
seite 1 1
zahnrad
sh mitte
re 55 schr16tel 74*2 li 55
aufkurs 90-8
sa
seite 2 2
zahnrad
sh mitte
re 55 schr16tel 74*3 li 55
aufkurs 90-11
sa
seite 3 3
zahnrad

@setflip 0 2

sh aufxy -10 (-10) sa

solange ~taste? [@autoflip]
```

Abb. 5.8.4 Ein sich drehendes Zahnrad wird dargestellt

Aufgabe:

Ändern Sie das Programm so, daß auch führende Nullen mit ausgegeben werden!

Ein weiteres Programm zeigt *Abb. 5.8.2*. Es hat die Aufgabe, die Ausgänge der IOE-Baugruppe zu testen. Dazu wird einfach laufend der Wert 0 bis 255 ausgegeben. Mit einem Oszilloskop kann man den Wechsel sehen. Wer kein Skop zur Verfügung hat, kann das Signal auch mit einem Prüfstift auswerten. Durch Drücken einer Taste läßt sich das Programm abbrechen. Dazu steht am Anfang die „WENN"-Abfrage. Dann wird mit dem „PORT"-Befehl der Wert „N" an den Port mit der Adresse $30 ausgegeben. Schließlich wird das Programm rekursiv aufgerufen, diesmal aber mit einem neuen „N". Hier ist nun ein besonderer Trick mit dabei. Es sollen ja nur Werte zwischen 0 und 255 ausgegeben werden. Zunächst wird die Variable „N" um eins erhöht. Dann wird das Ergebnis aber mit dem Wert $FF, also 255 und verknüpft. Dadurch folgt nach 255 nicht der Wert 256, sondern wieder der Wert 0.

Wie wir bereits wissen, kann man in Gosi die Unterprogramme des Grundprogramms aufrufen, so z. B. die A/D-Umsetzerroutine. *Abb. 5.8.3* zeigt ein Programmbeispiel. Das Unterprogramm „HOLEAD" ruft das Unterprogramm im Grundprogramm auf. Das Ergebnis der A/D-Umsetzung, ein Wert zwischen 0 und 1023, liegt im Register D0.L. Damit kann man das Unterprogramm wie eine Funktion aufrufen, ohne daß ein „RG"-Befehl nötig ist. Im Hauptprogramm wird die Funktion „HOLEAD" eingesetzt. Zunächst wird eine Linie mit der alten Position „POS" gelöscht. Dazu wird der Befehl „STIFT 1" verwendet, der den Schreibstift auf „löschen" umschaltet. „POS" gibt dabei den absoluten Winkel der Schildkröte an. Dann wird der A/D-Wert berechnet und der Variablen „POS" zugewiesen. Mit „STIFT 0" wird der Schreibstift auf „schreiben" geschaltet und die neue Linie im neuen Winkel ausgeben. Danach wird das Programm wiederholt, bis man eine Taste drückt.

Wenn man an den A/D-Umsetzer ein Potentiometer anschließt, so folgt der Zeiger auf dem Bildschirm der Drehung des Potentiometers.

Aufgaben:

1. Verbessern Sie das Programm, indem Sie noch einen kleinen Zeiger hinzufügen, der den 10ten Teil des großen Zeigers angibt, dadurch wird die Ausgabe eindeutig. Denn der A/D-Umsetzer liefert Werte zwischen 0 und 1023, der Zeiger kehrt aber alle 360° an die gleiche Position zurück.
2. Verbessern Sie das Programm, indem Sie mehrere Bildschirmseiten verwenden! Dazu kann man den Befehl „SEITE schreib lese" verwenden.

Abb. 5.8.4 zeigt ein Programm für ein sich drehendes Zahnrad, das Ihnen als Anregung für die Seitenumschaltung dienen soll. Analysieren Sie das Programm, und übertragen Sie die Erkenntnisse auf das Zeigerinstrument! Unbekannte Befehle können Sie im Anhang nachsehen.

Abb. 5.8.5 zeigt ein Programm, mit dem man die D/A-Umsetzer-Baugruppe ansprechen kann. Schaltet man den y-Eingang eines Oszilloskops an den Kanal 0 und den x-

Eingang an den Kanal 1, so kann man eine Ellipse auf dem Skopschirm sehen. An den Ausgängen der beiden D/A-Umsetzer liegt eine phasenverschobene Sinus-Spannung. Es entsteht somit auf einfache Weise ein computergesteuerter Funktionsgenerator. Man muß nur die Funktion berechnen und über die D/A-Umsetzer ausgeben. Dank der Sprache Gosi ist das Programm auch schnell genug. Es wäre in dieser Disziplin nur noch durch ein Assembler-Programm geringfügig zu übertreffen.

6 Betriebssysteme und Disketten-Laufwerke

Bisher hatten wir zur Abspeicherung unserer Programme und Daten einen Cassettenrekorder verwendet. Dies ist zwar eine brauchbare und preiswerte Möglichkeit, es gibt aber elegantere Einrichtungen.

In der Industrie gibt es einmal die sogenannten Bandgeräte, wie sie bei Großrechnern eingesetzt werden. Außer diesen gibt es dann noch Plattenstationen. Diese Geräte besitzen eine hohe Speicherkapazität und sind sehr schnell. Insbesondere die Geschwindigkeit würde uns natürlich interessieren. Beim Cassettenrekorder haben wir eine Aufzeichnungsrate von 1200 Baud gehabt, das sind etwa 100 Zeichen pro Sekunde.

Doch die Geräte, die für Großrechner entworfen wurden, sind viel zu teuer und zu unhandlich, um für unseren Rechner geeignet zu sein. Für Mikrocomputer gibt es solche Geräte gewissermaßen in Miniaturausführung. Zum einen gibt es kleinere Bandlaufwerke, die sehr schnell sind, und ebenso kleinere Plattenlaufwerke. Mit Bandlaufwerken wollen wir uns nicht mehr befassen, aber mit den kleinen Plattenlaufwerken.

Dabei unterscheidet man zwei Grundsätzliche Arten. Einmal die sogenannten Disketten- und dann die Festplatten-Laufwerke. Disketten-Laufwerke haben eine geringere Speicherkapazität als Festplatten, sind aber preiswerter als diese.

Zum Vergleich: Disketten-Laufwerke besitzen eine Kapazität zwischen ca. 70 KByte und 1,5 MByte, je nach Ausführung. Festplattenlaufwerke liegen zwischen 5 und 300 MByte.

Die Disketten-Laufwerke haben noch einen besonderen Vorteil. Man kann die Disketten austauschen. Die Disketten sind der eigentliche Träger der Information; sie bestehen aus einer magnetisierbar beschichteten Kunststoffscheibe. Um dieser biegsamen Kunststoffscheibe mehr Stabilität zu verleihen und um sie zu schützen, ist sie in einer Hülle untergebracht. Die Biegsamkeit gibt dieser Scheibe auch den Namen: „flexible disc" oder Floppy.

Bei einer Festplatte befindet sich eine beschichtete Metallscheibe fest montiert im Inneren des Laufwerks. Durch die höhere Festigkeit und noch einige Besonderheiten ist es möglich, mehr Information unterzubringen. Es gibt zwar auch Laufwerke, bei denen man die „feste Platte" austauschen kann, diese sind jedoch noch nicht sehr verbreitet.

An den NDR-Computer kann man natürlich auch eine Festplatte anschließen, dazu gibt es eine eigene Baugruppe. Für „normale" Anwendungen genügt eine Diskette aber völlig. Wir widmen uns in diesem Kapitel dem Disketten-Laufwerk.

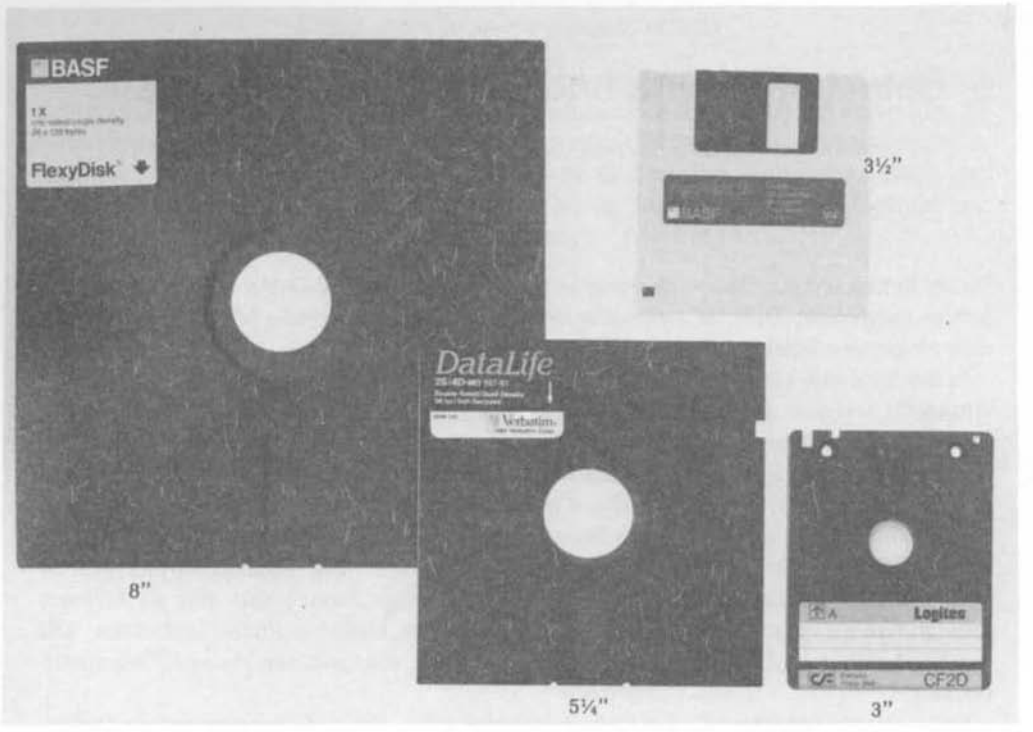


Abb. 6.1.1 Verschiedene Disketten-Formate

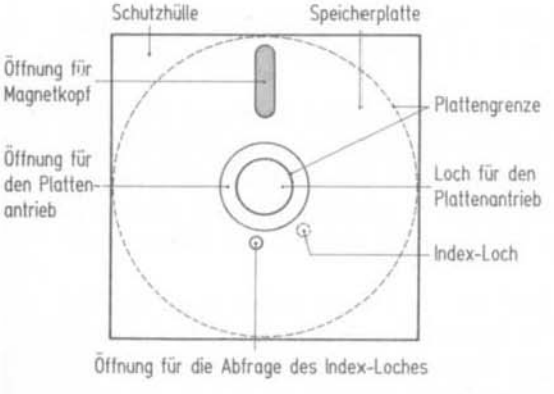


Abb. 6.1.2
Aufbau einer Diskette

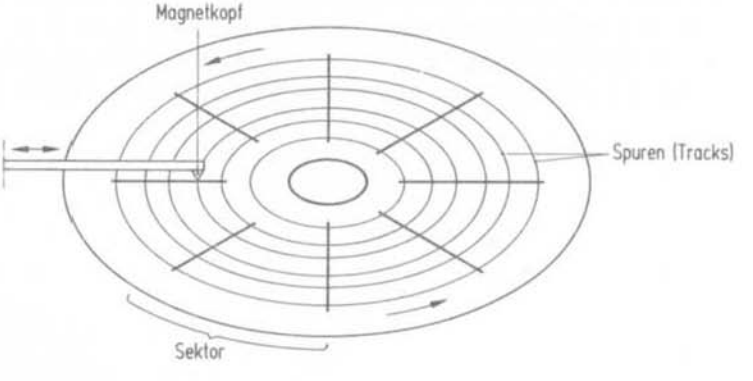


Abb. 6.1.3
Die Anordnung der Spuren und Sektoren auf einer Diskette

6.1 Laufwerke und Formate

Alles wäre ganz einfach, wenn es nicht so viele unterschiedliche Arten von Laufwerken gäbe. Die ersten Disketten-Laufwerke verwendeten sogenannte 8"-Disketten. Die Dimensionierung in Zoll wurde gewählt, da diese Technik aus dem Amerikanischen kam.

Die 8"-Laufwerke haben die größten Maße. Später kamen dann die 5¼"-Laufwerke hinzu. Sie verwenden eine kleinere Diskette, nämlich eine mit 5¼" Durchmesser.

Diese 5¼" Disketten-Laufwerke sind zwar heute am weitesten verbreitet, aber schon zeigt sich eine weitere Neuerung. Die 3½"-Diskette und die 3"-Diskette. *Abb. 6.1.1* zeigt eine Zusammenstellung von unterschiedlichen Disketten.

Die 3½" und die 3"-Diskette unterscheiden sich nicht nur in der Größe, sondern auch in der Verpackkungsart vom 5¼"-Format. Nun dient nicht mehr einfach eine Hülle als Schutz, sondern ein knickfestes Kunststoffgehäuse. Dadurch sind diese Disketten sehr robust.

Die 8"-Disketten werden auch Maxi-Disketten genannt, die 5¼"-Floppies heißen Mini-Disketten und die 3½"- sowie die 3"-Disketten nennt man Mikro-Disketten.

Im Prinzip ist die Arbeitsweise aller Disketten die gleiche. *Abb. 6.1.2* zeigt den schematischen Aufbau einer Diskette. Ein großes Mittelloch dient dem Antrieb. Ein Motor im Disketten-Laufwerk versetzt die Magnetscheibe im Inneren der Hülle in Drehung.

Eine weitere Öffnung in der Diskettenhülle ist für den Magnetkopf vorgesehen. Dieser Magnetkopf kann sich entlang dieses Schlitzes bewegen. Der Antrieb erfolgt meist über einen Schrittmotor. Der Magnetkopf dient wie bei einem Tonbandgerät der Aufzeichnung und Wiedergabe von magnetischen Schwingungen.

In der Nähe des Mittelloches befinden sich ein oder zwei kleinere Löcher für den sogenannten Index; dieser wird für eine Positionserkennung benötigt.

Manchmal, meist bei Mini-Disketten (5¼"), befindet sich an der Seite noch ein weiterer Schlitz. Der dient dem Schreibschutz. Wenn man ihn bei Mini-Disketten zuklebt, so kann man keine Daten auf die Diskette schreiben. Bei Maxi-Disketten (8") ist es genau umgekehrt, ist der Schlitz offen, so kann man keine Daten schreiben. Bei den Mikro-Disketten ist ebenfalls ein Schreibschutz vorhanden, meist ein kleiner Kunststoffschieber, der fest in die Hülle eingebaut ist.

Um die Lage des Schreibschutzes zu prüfen, befindet sich im Laufwerk ein kleiner Schalter oder eine Lichtschranke, mit der man feststellen kann, ob der Schlitz zu oder offen ist. Die Schreibschutzeinrichtung dient dazu, vor unbeabsichtigtem Ändern wichtiger Daten zu schützen. Wie wir bereits hörten, wird die in das Laufwerk eingelegte Diskette in Rotation versetzt. Nun kann man mit dem Magnetkopf auf die Information zugreifen. *Abb. 6.1.3* zeigt das Schema. Ist der Magnetkopf in einer festen Position fixiert, so kann er auf eine ringförmige Spur (Track) zugreifen. Durch das Verschieben des Kopfes kann man eine andere Spur erreichen. Wichtig ist dabei, daß die Daten nicht wie bei der Schallplatte spiralförmig aufgezeichnet werden, sondern ringförmig. Der Magnetkopf bewegt sich also nur, um auf die Spur zu gelangen. Er bleibt dagegen fest, während man Daten auf die Spur bringt oder von der Spur liest. Das Positionieren geschieht dabei meist mit Hilfe eines Schrittmotors.

Die Diskette ist also in Spuren eingeteilt, deren Anzahl nicht einheitlich ist. Sie wird u.a. durch die Größe der Diskette festgelegt. So kann man bei einer Diskette mit einem großen Durchmesser z. B. 77 Spuren unterbringen, während bei einer Mini-Diskette mit gleichem Spurabstand nur 40 Spuren möglich sind. Moderne Minilaufwerke arbeiten mit besseren Magnetköpfen und können zum Beispiel 80 Spuren auf die Mini-Diskette bringen, indem sie den halben Spurabstand verwenden.

Dann gibt es noch Laufwerke, die auch die Rückseite der Diskette beschreiben können und dazu auf der Rückseite einen zweiten Magnetkopf besitzen. Damit kann man natürlich die doppelte Kapazität erreichen.

Nun ein wichtiger Punkt. Zunächst einmal würde es genügen, mit der Spureinteilung zu arbeiten. Nachdem aber eine Spur bereits z. B. 5 KByte fassen kann, ist diese Einheit zu groß; man würde bei kleinen Datenteilen viel Kapazität verlieren. Besser ist es, wenn man mit kleineren Informationsblöcken arbeitet. Ein gängiges Maß sind z. B. 128 Byte-Pakete. Man nennt diese Aufteiltung Sektoren. Ein Sektor ist die kleinste Informationseinheit, an die man bei einem Diskettenlaufwerk herankommt. Warum?
Nun, wollte man ein einzelnes Byte schreiben, so müßte man ganz exakt die Stelle unter vielen Bytes finden, an der der Magnetkopf das einzelne Byte einfügen soll. Das geht wegen der ungenauen Mechanik und auch wegen physikalischer Eigenschaften nicht. Daher ist es auch nötig, die einzelnen Sektoren durch einen kleinen Sicherheitsabstand voneinander zu trennen. Damit man später einen bestimmten Sektor auf der Spur wiederfindet, sind die Sektoren numeriert. Üblicherweise fängt man bei Sektoren mit der Zahl 1 an. Bei Mini-Disketten können z.B. Sektoren von 1 bis 16 mit je 256 Byte pro Sektor oder Sektoren von 1 bis 5 mit 1024 Byte pro Sektor abgespeichert werden. Wieviele Byte ein Sektor faßt ist unterschiedlich und hängt sehr stark von der Anwendung ab. Man kann zwischen 128, 256, 512 und 1024 wählen. Je größer die Sektoren sind, desto weniger getrennte Datenpakete stehen zur Verfügung. Wird ein Sektor zerstört, so ist um so mehr Information verloren. Umgekehrt bekommt man um so mehr Information auf die Diskette, da weniger Platz für die Zwischenräume zwischen den einzelnen Sektoren verloren geht. Die Spuren werden, beginnend bei Spur 0, numeriert.

Bei 8"-Laufwerken sind meist 77 Spuren vorhanden (0 bis 76), bei Minilaufwerken gibt es drei wichtige Größen: 35 Spuren (0 bis 34), 40 Spuren (0 bis 39) und 80 Spuren (0 bis 79).

Wie erkennt man, welcher Sektor gerade dran ist? Dazu gibt es zwei Möglichkeiten. Entweder schreibt man die Sektornummer unmittelbar als kleinen Datenblock vor den Sektor, oder man verwendet Löcher in der Diskette, die man abzählen kann. Das erste Verfahren nennt man Soft-, das zweite Hardsektorierung. Disketten mit hardsektorierten Sektoren sind heute fast nicht mehr üblich, denn die Sektoreinteilung ist dabei starr vorgegeben. Ein Relikt aus dieser Zeit ist aber das sogenannte Indexloch. Es gibt den Start einer Spur an. Man benötigt es aber eigentlich nur noch, wenn man zum ersten Mal Sektoren auf die Diskette schreibt, um zu garantieren, daß der erste Sektor immer an der gleichen Stelle beginnt. Bei hardsektorierten Disketten ist das Indexloch zwischen den Sektorlöchern angeordnet.

Wie sieht nun der Aufbau eines Laufwerks aus? Ein Antriebsmotor (drive motor) sorgt für die Rotation der Diskettenscheibe. Der Motor wird bei Minilaufwerken meist mit +12 V betrieben, bei 8"-Laufwerken entweder mit +24 V oder mit Netzspannung.

Dann gibt es einen Schrittmotor (Stepper motor), der den Magnetkopf positioniert. Der Schrittmotor benötigt entweder +12 V (bei Minilaufwerken) oder +24 V (bei 8"-Laufwerken). Über eine Steuerlogik (control-logic) wird er bedient. Von außen gelangen an das Laufwerk, das die Steuerelektronik mit eingebaut hat, die Leitungen STEP und DIRECTION, die den Schrittmotor bewegen können. Damit der Rechner feststellen kann, wann der Magnetkopf über der Spur 0 liegt, gibt es einen „Track 0 LED/Detector", der aus einer Lichtschranke oder einem Schalter besteht und mit dem Arm des Magnetkopfes gekoppelt ist.

Ferner gibt es einen „write protect detector", der prüft, ob der Schreibschlitz abgeklebt oder offen ist, und einen „index detector" für das Indexloch. Über eine Schreib- und Leselogik kann auf den Magnetkopf zugegriffen werden.

Der Disketten-Controller, auch Floppy-Controller genannt, ein integrierter Schaltkreis, der mit dem Mikrocomputer zusammenarbeitet, sorgt dann dafür, daß alle Signale, die das Laufwerk braucht, erzeugt und verarbeitet werden.

Aufzeichnungsverfahren

Derjenige Leser, der nun weiter ins Detail gehen will, kann an dieser Stelle weiterlesen. Wer dies nicht will, kann gleich zum Abschnitt 6.2 springen.

Die Daten werden sequenziell und in einem speziellen Format vom Computer zum Disketten-Laufwerk übertragen. Man kann nämlich nicht einfach Einsen und Nullen auf der Diskette aufzeichnen. Da sich die Diskette bei der Wiedergabe nie mit exakt der gleichen Geschwindigkeit dreht, ist es nötig, eine Information über den Datentakt mit zu übertragen.

1. *Das FM-Verfahren*

Wurde dieser Datentakt mit aufgezeichnet, kann man bei der Wiedergabe die Daten in ein Schieberegister laden und dieses mit dem zurückgewonnenen Takt betreiben. Die einfachste und älteste Methode ist das FM-Verfahren, das auch als Single-Density-Aufzeichnung oder Aufzeichnung mit einfacher Schreibdichte bekannt geworden ist. *Abb. 6.1.4* zeigt das Schema. FM ist die Abkürzung für Frequenz-Modulation.

Die Daten werden bei FM z. B. im Abstand von 4 µs übertragen. Dieses gilt für das 8"-Aufzeichnungsverfahren; die meisten Mini- und Mikro-Laufwerke arbeiten bei FM mit der halben Übertragungsrate, also 8 µs.

Zu Beginn einer jeden Bitzelle wird zunächst ein Taktimpuls erzeugt. Er ist in der Abb. mit „T" markiert. Wenn der Inhalt der Bitzelle 1 sein soll, so wird nach 2 µs ein weiterer Puls übertragen. Die einzelnen Pulse sind ca. 200 ns breit. Ist die Bitzelle 0, so bleibt der Puls aus. Der Datenpuls für das 1-Signal ist in der Abb. mit „D" gekennzeichnet.

Den Takt kann man später durch ein Monoflop oder eine sogenannte PLL- (phase locked loop) Schaltung wieder zurückgewinnen. Bei jedem Puls wird auf der Diskette

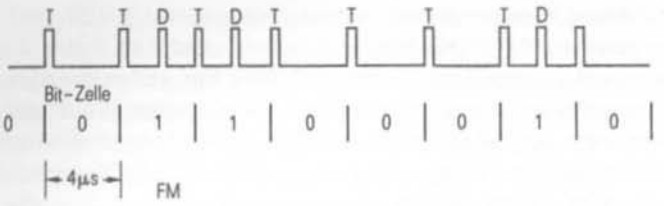


Abb. 6.1.4
Die FM-Aufzeichnung

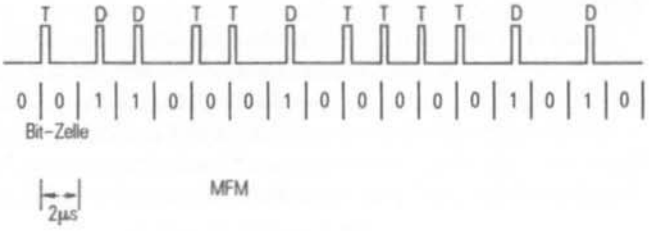


Abb. 6.1.5
Die MFM-Aufzeichnung

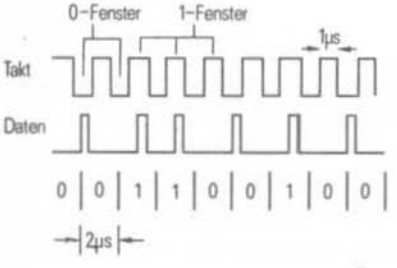


Abb. 6.1.6
Die Rückgewinnung von Daten bei MFM

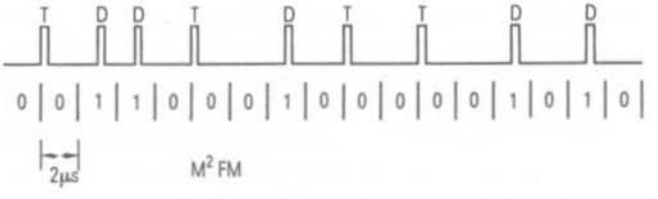


Abb. 6.1.7
Das M²-FM-Prinzip

Tabelle 6.1.1
Die GCR-Umrechnungs-Tabelle

4-Bit-Gruppen der Daten	5-Bit-Kombinationen der Aufzeichnung
0000	11001
0001	11011
0010	10010
0011	10011
0100	11101
0101	10101
0110	10110
0111	10111
1000	11010
1001	01001
1010	01010
1011	01011
1100	11110
1101	01101
1110	01110
1111	01111

durch den Magnetkopf ein Flußwechsel erzeugt. Bei der Wiedergabe kann eine auf dem Disketten-Laufwerk vorhandene Schaltung wieder die ursprünglichen Pulse ableiten.

Zum Rückgewinnen des Taktes hier ein Beispiel. Wir triggern bei jedem Puls ein Monoflop, dessen Zeitdauer auf 3 µs eingestellt ist. Dann wird dieses Monoflop automatisch den Takt wiederherstellen. Das Monoflop wird beim T-Puls ausgelöst. Es fällt kurz vor dem nächsten T-Puls wieder in die 0-Lage zurück und wird dann vom nächsten T-Puls erneut getriggert es darf natürlich nicht retriggerbar sein. Pulse, die innerhalb der ausgelösten Monoflop-Dauer ankommen, bleiben somit unbeachtet. Nun kann es noch vorkommen, daß das Monoflop durch einen D-Puls getriggert wurde, denn woher sollte das Monoflop wissen, wann der Magnetkopf beginnt, die Daten zu lesen. Das Monoflop wird sich dann erst nach einem 0-Signal auf die T-Pulse einstellen. Es rastet, wie man sagt, erst dann auf den Takt ein. Daher benötigt man auf der Diskette sogenannte Synchronisations-Daten, die den eigentlichen Daten vorangestellt sind. Ferner muß man auch irgendwoher wissen, wo ein Byte, bzw. ein Sektor anfängt. Auch dafür gibt es spezielle Datenmuster, wie wir noch sehen werden.

2. Das MFM-Verfahren

Doch nun zu einem moderneren Aufzeichnungsverfahren. Bei FM geht viel Zeit dadurch verloren, daß der Takt bei jedem Bit mitaufgezeichnet werden muß. Beim MFM-Verfahren ist dies nicht notwendig. MFM ist die Abkürzung für modified frequency Modulation. Es erlaubt die doppelte Aufzeichnungsdichte, ohne daß die Anzahl der Flußwechsel wesentlich erhöht wird. *Abb. 6.1.5* zeigt ein Beispiel. Die Bitzelle ist nur noch halb so groß, nämlich 2 µs bei 8″-MFM, oder 4µs bei den meisten Mini- und Mikrodisketten-Laufwerken.

Es gilt nun folgende Regel zur Erzeugung der Pulse: Bei einer Zelle mit einem 1-Bit-Signal wird ein Puls in der Mitte der Bitzelle übertragen; bei einer Bitzelle mit einem 0-Bit-Signal wird am Anfang der Bitzelle ein Puls übertragen, wenn keine Bitzelle mit einem 1-Signal davor lag. Der Puls, der bei der Bitzelle mit einem 0-Signal übertragen wird, ist der Takt.

Nun bleibt die Frage, wie bekommt man daraus jemals wieder die Original-Information zurück? *Abb. 6.1.6* zeigt eine Möglichkeit: Ein Takt wird mit Hilfe einer PLL-Schaltung aus dem Datenstrom gewonnen. Jede Takthälfte, also die 1-Dauer und die 0-Dauer, dient als Fenster für das Datensignal. Für die Erkennung von 1 und 0 bleibt bei MFM jetzt nur noch die halbe Zeit, was die Empfindlichkeit des Verfahres erhöht. Daher kann man auch nur solche Laufwerke für MFM verwenden, die dafür vom Hersteller auch als geeignet spezifiziert wurden. Dies sind aber die meisten heute auf dem Markt befindlichen.

Eine weitere Eigenschaft, die erst bei MFM zu tragen kommt, ist die folgende. Zwei aufgezeichnete Pulse tendieren bei der Wiedergabe dazu, sich voneinander zu entfernen, bzw. entfernt zu erscheinen. Dies geschieht zum Teil wegen Überlagerungseffekten auf der Magnetschicht.

Man kann diesen Effekt vermindern, indem man die Pulse, die dicht beieinanderliegen, bei der Aufzeichnung künstlich dichter zusammenrückt, so daß sie nachher normalen Abstand haben. Diese Pulse werden dabei bei der Aufzeichnung um ca. 150 ns versetzt. Der Wert ist laufwerksabhängig und tritt insbesondere bei den inneren Spuren in Erscheinung, denn dort ist die Aufzeichnungsdichte höher als außen. Diese Pulszusammenrückung nennt man auch „Precompensation".

3. *Das M²FM-Verfahren und GCR*

Seltener verwendet werden diese beiden Verfahren, die hier nur der Vollständigkeit halber genannt seien. Standard-Controller-ICs beherrschen sie nämlich nicht.

Abb. 6.1.7 zeigt ein Schema zum M²FM. Wie bei MFM wird für das 1-Signal einer Bitzelle in der Zellenmitte ein Puls aufgezeichnet. Bei einem 0-Signal wird nur dann ein Puls am Anfang der Zelle aufgezeichnet, wenn die vorherige Bitzelle, unabhängig davon ob 0- oder 1-Signal, keinen Puls enthielt.

Durch diese Regel ergibt sich im Mittel ein etwas weiterer Abstand der Pulse, was eine Precompensation meist unnötig macht. Dies ist der Vorteil des Verfahrens. Die Rückgewinnung gestaltet sich aber deutlich komplizierter als bei MFM, daher hat es sich wohl auch gegenüber MFM nicht durchgesetzt.

Nun gibt es noch ein weiteres Verfahren, das GCR (Group Code Recording) genannt wird. Dabei werden jeweils vier Datenbits über eine Tabelle auf fünf abgelieferte Datenbits umgerechnet, wie in *Tabelle 6.1.1* zu sehen ist. Der Code dieser 5-Bit-Gruppen ist so gewählt, daß nie mehr als zwei Nullen aufeinanderfolgen. Bei dieser Kombination würde kein Puls aufgezeichnet werden. Das Signal eignet sich nämlich dann zur direkten Aufzeichnung, und man kann immer dann einen Puls erzeugen, wenn ein 1-Signal im 5-Bit-Code steht. Durch die besondere Eigenschaft, nie mehr als zwei Nullen hintereinander stehen zu haben, ist garaniert, daß man den Takt einfach aus den vorhandenen Daten zurückgewinnen kann.

4. *Die Adreßmarken*

Wie schon gesagt, kann man die Daten nicht einfach unverändert auf die Diskette aufzeichnen, sondern es sind Kennungen für den Start des Datenfeldes und Informationen, welcher Sektor gerade angesprochen wird, erforderlich. *Abb. 6.1.8* zeigt ein Schema. Zunächst folgt nach dem Index-Loch im Abstand von etwa 46 Bytes bei FM die sogenannte Index-Adreß-Marke. Damit ist es dem Floppy-Controller möglich, den Start nach dem Index-Puls festzustellen. Allerdings wird diese Index-Marke bei den meinsten Systemen nicht mehr benötigt. Danach folgt eine Lücke, in der 32 Füllbytes bei FM liegen, und dann die sogenannte ID-Adreß-Marke. Danach folgt ein Feld, das Auskunft über Sektor, Spur, Diskettenseite und Byteanzahl des nachfolgenden Datenfeldes gibt.

Damit kann der Disketten-Controller feststellen, ob der richtige Sektor gelesen wird, und, als zusätzliche Kontrolle, ob Spur und Seite richtig angesprochen sind. Danach folgen die CRC-Bytes, mit denen wir uns später ausführlicher beschäftigen werden. Sie

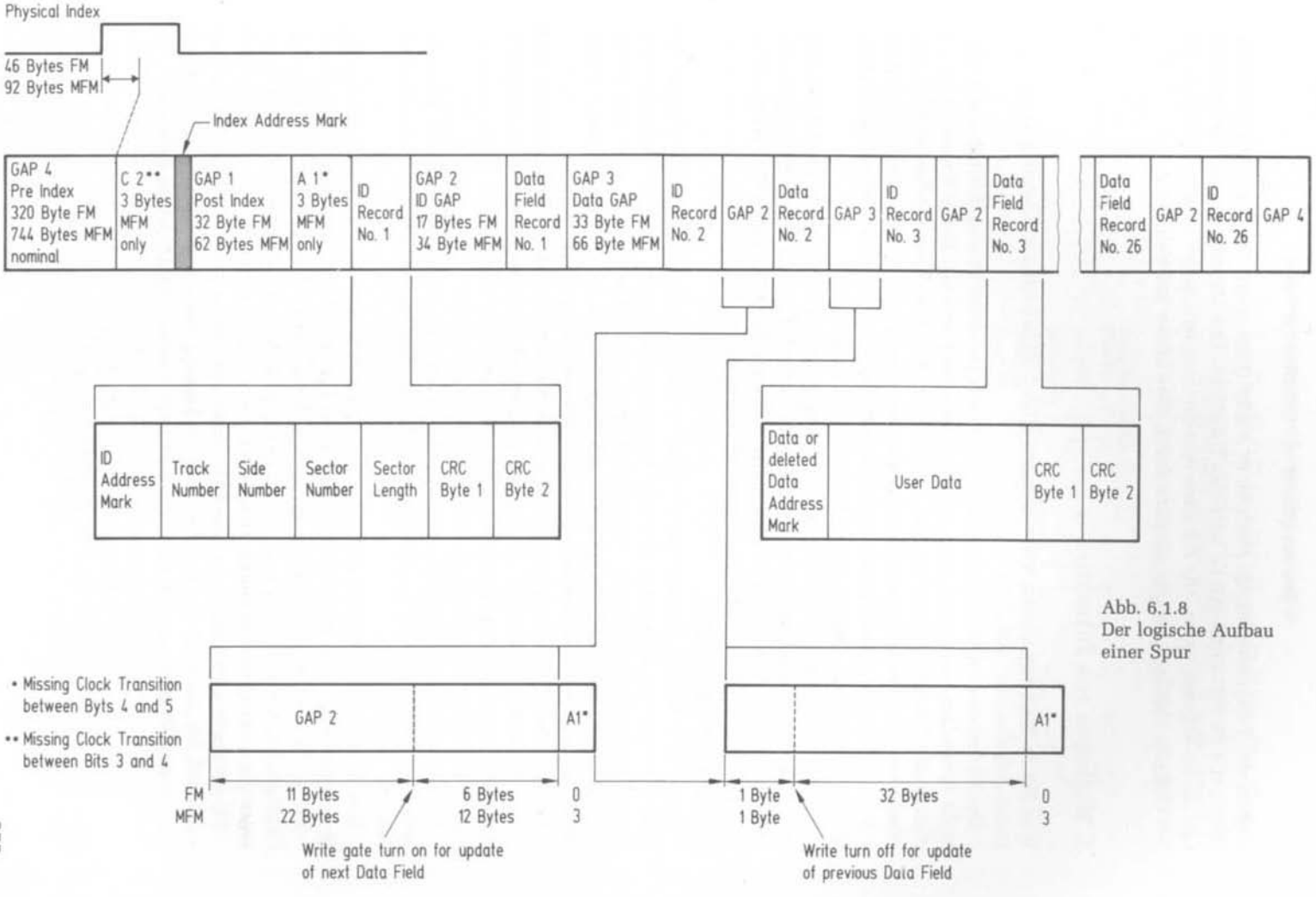


Abb. 6.1.8
Der logische Aufbau einer Spur

dienen der Fehlererkennung. Nach einer Lücke (engl. gap), die mit Bytes gefüllt ist, folgen die Daten-Adreß-Marke und das Datenfeld. Das Datenfeld wird ebenfalls mit CRC-Bytes abgeschlossen. Die Sektoren werden dann in dieser Weise nacheinander auf der Diskette abgelegt, jeweils getrennt durch eine Lücke mit Füll-Bytes.

5. *Weglassen von Taktpulsen*

Was hat es nun mit diesen Markierungen (Daten-Adreß-Marke, ID-Adreß-Marke und Index-Adreßmarke) auf sich?

Der Disketten-Controller kann allein durch das bisherige Datenaufzeichnungsformat nicht feststellen, wann ein Byte oder ein Datenfeld anfängt. Daher wurde dafür ein eigenes Bit-Format gewählt, bei dem auch die Taktbits eine Rolle spielen. Es gilt durch Weglassen von Taktpulsen ein Datenmuster zu erzeugen, das sonst nie als Bitmuster auftritt. Eine spezielle Erkennungslogik im Disketten-Controller filtert dann diese Stellen heraus und synchronisiert die restliche Logik darauf.

Nun zu den einzelnen Marken. *Abb. 6.1.9* zeigt die Index-Adreß-Marke. Dort wurden zwei Takt-Bits weggelassen. Das Datenmuster lautet $FC. Auch das Taktmuster besitzt nun einen Code, nämlich $D7. *Abb. 6.1.10* zeigt die ID-Adreß-Marke: Der Datenteil ist $FE, der Taktteil $C7. *Abb. 6.1.11* zeigt die Daten-Adreß-Marke; dort ist das Datenbyte $FB und der Taktteil $C7.

Ferner gibt es noch die sogenannte Deleted-Adreß-Marke, die *Abb. 6.1.12* zeigt. Sie kann anstelle der Daten-Adreß-Marke stehen und bedeutet, daß die nachfolgenden Daten ungültig sind. Dieses Verfahren, Daten zu löschen, wird aber nur selten gebraucht. Es findet z.B. bei IBM-Anlagen mit einer IBM 3740-Aufzeichnung Anwendung. Bei dieser Adreß-Marke sind Data = $F8 und Takt = $C7.

Bei dem MFM-Aufzeichnungsverfahren ist die Sache nicht so einfach, denn dort fehlen ja bereits Taktpulse durch das dichtere Aufzeichnungsverfahren. Dennoch finden sich zwei Muster, bei denen nochmals Taktpulse weggelassen werden können und so eine Unterscheidung zu normalen Daten bringen:

Einmal das Muster $A1, das in *Abb. 6.1.13* zu sehen ist. Dort fehlt der Taktpuls zwischen den Bits 4 und 5, und dann das Muster $C2, bei dem der Taktpuls zwischen den Bits 3 und 4 fehlt.

Das Muster $C2 wird nun dreimal hintereinander aufgezeichnet, und dann folgt die Index-Marke mit dem Code $FC, und zwar als Datenwert ohne fehlende Taktpulse.

Der Code $A1 wird ebenfalls dreimal aufgezeichnet, danach folgt entweder der Datenwert $FE als ID-Adreß-Marke oder $FB als Data-Adreß-Marke, jeweils mit Taktpulsen wie bei einem normalen Datenwert.

Bliebe noch zu klären, wieviele Byte in einem Sektor stehen. Dafür gibt es eine Vorschrift für das Feld im ID-Adreß-Teil. So können nur vier verschiedene Sektorlängen verwendet werden: 128, 256, 512 und 1024 Byte pro Sektor, siehe *Tab. 6.1.2*

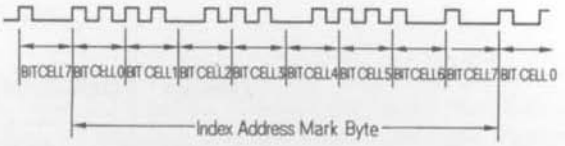
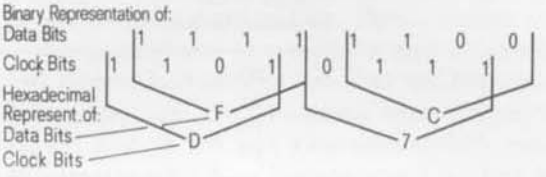


Abb. 6.1.9
Index-Adreß-Marke



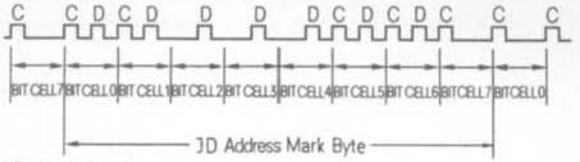


Abb. 6.1.10
ID-Adreß-Marke

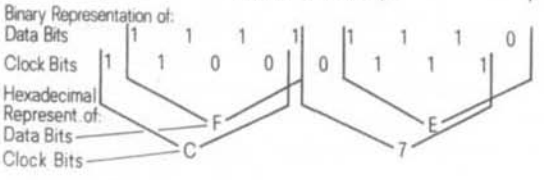
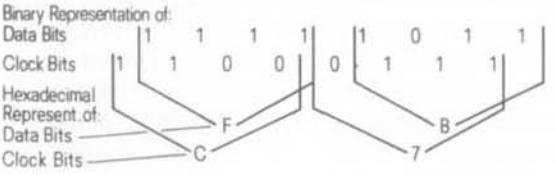


Abb. 6.1.11
Data-Adreß-Marke



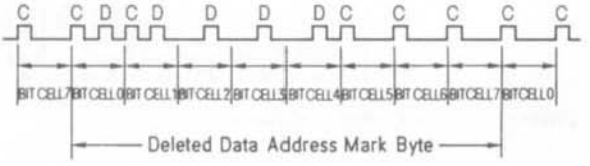


Abb. 6.1.12
Deleted-Data-Adreß-Marke

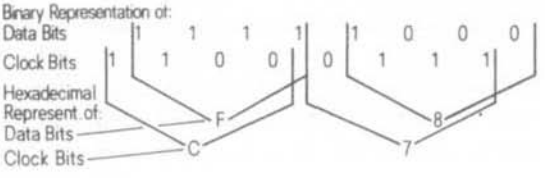


Tabelle 6.1.2 Code-Zuordnung

Feld in der ID-Adreß-Marke	Sektorlänge (Bytes)
00	128
01	256
02	512
03	1024

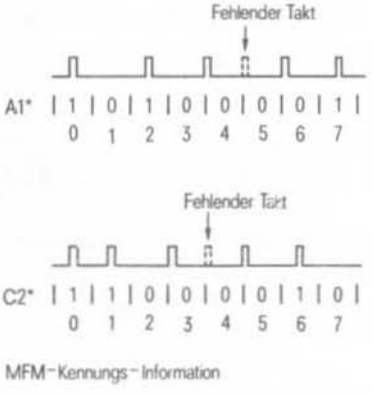


Abb. 6.1.13 MFM-Kennungs-Information

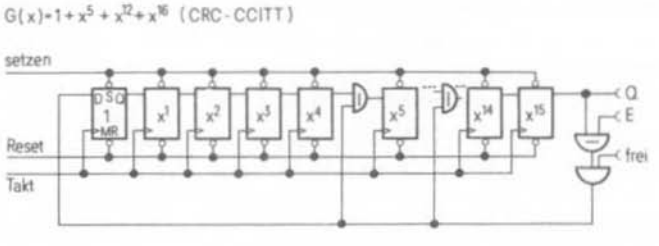


Abb. 6.1.14 So könnte man die CRC-Informationen erzeugen

6. *Fehlererkennung mit CRC-Bytes*

Das Geheimnis des CRC-Feldes soll nun gelüftet werden. CRC ist die Abkürzung für Cyclic Redundancy Check. Durch diese Information erhalten die Daten eine Redundanz, die es ermöglicht, festzustellen, ob einzelne Bits fehlerhaft übertragen wurden. Die Erzeugung der CRC-Information ist nicht ganz einfach. Darum wollen wir uns zunächst mit einer einfacheren Form beschäftigen, so wie sie für unsere Cassettenaufzeichnung Anwendung findet. Bei diesem Prüfsummenverfahren werden einfach bei der Aufzeichnung alle Daten aufsummiert und die Summe als letzter Datenwert mit abgespeichert. Dabei verwenden wir hier 16 Bit und vergessen einfach einen ggf. vorhandenen Übertrag. Bei der Wiedergabe wird diese Summe erneut gebildet und mit den letzten beiden Bytes verglichen. Stimmt die Summe überein, war kein Fehler vorhanden, sonst wird ein Fehler erkannt. Nun können sich zwei Fehler aber gegeneinander aufheben. Beispiel:
00 01 02 03
wurde gesendet, die Summe ist 06
00 03 00 03
wurde empfangen, die Summe ist ebenfalls 06. Die Prüfsummen-Methode ist fehlgeschlagen. Bei der Berechnung von CRC wird eine aufwendigere Form der Berechnung verwendet, die sich sinnvoll nur durch eine Hardware-Lösung realisieren läßt. CRC hat die Eigenschaft, Ausfälle von Datenblöcken leichter zu erkennen als die einfache Prüfsumme. Daher wird bei Disketten-Aufzeichnungs-Verfahren die CRC-Methode bevorzugt.

Zur Berechnung wird ein sogenanntes Generator-Polynom definiert. Es gibt sehr unterschiedliche Formen; bei den gängigen Disketten-Controllern wird eine Form nach CCITT verwendet:

$G(X) = 1 + x^5 + x^{12} + x^{16}$.

Abb. 6.1.14 zeigt die dazugehörige Schaltung. Ein Reset-Eingang sorgt dafür, daß das Schieberegister auf einen definierten Wert gesetzt werden kann. Dann wird der Eingang „frei" auf 1 gelegt und zusammen mit einem Takt werden die Daten an „E" angelegt. Nach dem Ende des Datenstroms wird „frei" auf 0 gelegt, und die CRC-Bytes können aus dem Register herausgeschoben werden. Um nun einen Datenstrom zu testen, wird genauso wie vorher verfahren, nur daß nun auch die CRC-Bytes mit verrechnet werden. Das Ergebnis im Schieberegister muß anschließend 0 sein. Die Schieberegisterschaltung ist natürlich im Disketten-Controller fertig mit integriert.

7. *Das Formatieren*

Wie bekommt man nun all die Marken und Lückenfüller auf die Diskette? Dies geschieht beim Formatieren. Das Formatieren geschieht durch den Disketten-Controller, dafür sind spezielle Befehle vorhanden. Beim FD1797, wie er auf unserer FLO2-Baugruppe zu finden ist, gibt es zum Beispiel einen Befehl „Schreibe Track". Damit kann also eine ganze Spur auf einmal beschrieben werden. Nun genügt es aber nicht,

nur einfach Datenbytes aufzuzeichnen, sondern die speziellen Informationen für Adreß-Marken mit den fehlenden Taktpulsen und auch die CRC-Bytes müssen erzeugt werden. Dazu wurden beim Disketten-Controller einige Datenbytes reserviert und mit einer neuen Bedeutung versehen. *Tab. 6.1.3* zeigt die Zuordnung. $F7 zum Beispiel erzeugt die CRC-Bytes. Die Datenwerte $F5 bis $FF können damit natürlich nicht als Datenwerte in Datenfeldern beim Formatieren verwendet werden, was aber auch nicht weiter stört, denn beim Formatieren will man ja nur das Grundgerüst auf der Diskette ablegen. Meist wird der Wert $E5 als Datenwert in den Datenfeldern abgelegt.

Mit einem eigenen Schreib-Befehl kann man dann später beliebige Daten in den Datenfeldern ablegen.

Tab. 6.1.4 zeigt das Datenformat für den Spur-Schreib-Befehl bei der FM-Formatierung von 8″-Disketten und *Tabelle 6.1.5* das Format für die MFM-Datenfolge ebenfalls für das 8″-Format. Die Formatierung beginnt dabei nach der Erkennung des Index-Loches und dauert bis zum erneuten Durchlauf beim Index-Loch. Der Controller bricht die Aufzeichung dann automatisch ab.

8. Hersteller-Wirrwarr

Wie man sieht, gibt es sehr unterschiedliche Möglichkeiten, durch Variation von Sektorlänge und Anzahl der Sektoren verschiedene Formatierungen zu erreichen.

Bei 8″-Disketten hat man sich bisher zumindest bei der FM-Aufzeichnung weitgehendst an das IBM-3740-Format gehalten, bei MFM an das IBM-34-Format. Aber bei Mini- und Mikro-Disketten sieht das ganz anders aus. Dort gibt es zwar auch einen Standard namens ECMA-70, jedoch besitzt dieser keine gute Ausnützung der Diskettenkapazität. Daher haben sich die Hersteller von Computern jeweils ihr eigenes Format überlegt.

Hinzu kommt noch die unterschiedliche Anzahl von Spuren (35,40,77,80) und die Möglichkeit, einseitig oder doppelseitig zu arbeiten.

Wir werden bei unserem Computer zwei Formate bevorzugen, zum einen das 8″-Format, um Software aus Amerika beziehen zu können, und dann ein 5¼″-Format mit 800 KByte Speicherraum für doppelseitige Disketten mit 80 Spuren. Das Format kann auch bei den entsprechenden Mikrodisketten verwendet werden.

Im Prinzip können alle gängigen Laufwerke am NDR-KLEIN-Computer betrieben werden, man muß nur Konvertierprogramme dazu schreiben.

6.2 Lesen und Schreiben eines Sektors

In diesem Abschnitt werden wir den NDR-KLEIN-Computer um ein Disketten-Laufwerk ergänzen.

Sollte sich der Diskettenbetrieb anders verhalten als hier beschrieben, sehen Sie bitte im Kapitel 7, Abschnitt: „Die FLO2-Baugruppe" nach.

Das EPROM mit dem Programm „UFORM68K" ist im Anhang als Quell-Listing und als sedezimaler Ausdruck abgebildet. Das EPROM wird auf einen beliebigen, freien Platz

Tabelle 6.1.3 Reservierte Code für die Formatierung

DATA PATTERN IN DR (HEX)	FD179X INTERPRETATION IN FM ($\overline{DDEN}$ = 1)	FD1791/3 INTERPRETATION IN MFM ($\overline{DDEN}$ = 0)
00 thru F4	Write 00 thru F4 with CLK = FF	Write 00 thru F4, in MFM
F5	Not Allowed	Write A1* in MFM, Preset CRC
F6	Not Allowed	Write C2** in MFM
F7	Generate 2 CRC bytes	Generate 2 CRC bytes
F8 thru FB	Write F8 thru FB, Clk = C7, Preset CRC	Write F8 thru FB, in MFM
FC	Write FC with Clk = D7	Write FC in MFM
FD	Write FD with Clk = FF	Write FD in MFM
FE	Write FE, Clk = C7, Preset CRC	Write FE in MFM
FF	Write FF with Clk = FF	Write FF in MFM

* Missing clock transition between bits 4 and 5; ** Missing clock transition between bits 3 and 4

Tabelle 6.1.4 Erzeugung einer FM-Formatierung bei 8 Zoll

Anzahl (Bytes)	Daten
40	FF (oder 00)
6	00
1	FC (Indexmarke)
26	FF (oder 00)
6	00
1	FE (ID-Adreßmarke)
1	Spurnummer
1	Disk-Seite (00 oder 01)
1	Sektornummer (01 bis 1A)
1	00
1	F7 (zwei CRCs)
11	FF (oder 00)
6	00
1	FB (Daten-Adreßmarke)
128	Daten (bei IBM E5)
1	F7 (zwei CRCs)
27	FF (oder 00)
247	FF (oder 00) bis zum Indexloch

Tabelle 6.1.5 Erzeugung einer MFM-Formatierung bei 8 Zoll

Anzahl (Bytes)	Daten
80	4E
12	00
3	F6
1	FC (Index-Marke)
50	4E
12	00
3	F5
1	FE (ID-Adreßmarke)
1	Spurnummer (00 bis 4C)
1	Diskseite (00 oder 01)
1	Sektornummer (01 bis 1A)
1	01
1	F7 (zwei CRCs)
22	4E
12	00
3	F5
1	FB (Daten-Adreßmarke)
256	Daten
1	F7 (zwei CRCs)
54	4E
598	4E (bis zum Indexloch)

der ROA64-Baugruppe gesteckt. Es benötigt mindestens 16 KByte Arbeitsspeicher, also zwei RAMs auf der ROA64 mit dem Grundprogramm. Dieser RAM-Bereich muß bei der Adresse $9000 beginnen.

Bevor es an den Start geht, ein paar Hinweise zur Inbetriebnahme. Eine ausführliche Anleitung befindet sich in Kapitel 7. Das Disketten-Laufwerk benötigt eine Versorgungsspannung von +12 V und eine von +5 V. Die Stromstärke ist vom Laufwerkstyp abhängig, Sie sollten sich darum das Benutzer-Handbuch ansehen. Meist werden aber nicht mehr als 1 A pro Laufwerk benötigt.

Das Laufwerk wird über eine Flachbandleitung mit der FLO2-Baugruppe verbunden, *Abb. 6.2.1* zeigt ein Beispiel. Jedes Laufwerk besitzt ein paar Brücken zur Einstellung von diversen Optionen. Wichtig dabei ist die Laufwerksadresse. So wie wir früher Ports und Speicher adressiert haben, kann man auch Laufwerke adressieren. Dazu gibt es meist vier, manchmal auch nur drei Brücken. Diese Brücken sind mit D0, D1, D2 und D3 beschriftet, oder alternativ mit S1, S2, S3 und S4. Es darf immer nur eine Brücke gesteckt sein. Stecken Sie bitte die Brücke „D0", bzw. „S1". Wenn Sie ein zweites Laufwerk anschließen wollen, so kann man es zusätzlich an die Flachbandleitung anstecken. Dieses Laufwerk muß dann eine andere Adresse besitzen, also z.B. „D1" oder „S2". Auf jedem Laufwerk befindet sich normalerweise ein sogenanntes Widerstandsnetzwerk. Weder die Lage noch die Ausführung ist genormt, somit muß man es ebenfalls im Handbuch suchen. Wenn Sie zwei Laufwerke besitzen, dann muß bei einem von beiden dieses Widerstandsnetzwerk entfernt werden, sonst arbeiten die Laufwerke nicht einwandfrei. Das Widerstandsnetzwerk beinhaltet 330Ω 220Ω Widerstände, die als Abschluß für die Leitungen dienen. Zwei oder mehr solcher Widerstände würden für die FLO2-Baugruppe eine zu große Belastung bedeuten, sie könnte den nötigen Strom nicht mehr aufbringen. Diese Widerstandsnetzwerke sind immer auf einem Sockel montiert, *Abb. 6.2.2* zeigt Beispiele.

Wenn nun entsprechend dem Kapitel 7 die FLO2 eingestellt ist, so ist nun die Anlage betriebsbereit, und es kann losgehen.

Bevor man Daten lesen und schreiben kann, muß die Diskette formatiert werden. Dazu dient das Programm „UFORM68K", das in der Lage ist, unterschiedliche Diskettenformate zu erzeugen. Im Prinzip könnten Sie also auch andere Laufwerke oder Diskettenformate verwenden, wir beschränken uns jedoch hier auf ein Format, das NDR80-Format. Es arbeitet mit einer Kapazität von 800 KByte pro Diskette.

Formatieren

Das Programm „UFORM68K" wird mit Hilfe der Bibliotheksfunktion gestartet. *Abb. 6.2.3* zeigt alle Bildschirmmeldungen, die im Anschluß besprochen werden, in einer Übersicht. Zu Anfang wird uns die Möglichkeit geboten, sechs weitere Punkte aufzurufen. Der erste Punkt wählt Mini-Laufwerke aus; diese Einstellung ist auch für Mikro-Laufwerke (3½" und 3") zu verwenden.

Mit „2" werden Maxi-Laufwerke ausgewählt. Diese ersten beiden Punkte führen zu weiteren Unterpunkten, die es gestatten, sich das Diskettenformat selbst zusammenzustellen. Die Menue-Punkte „3" bis „5" hingegen wählen ganz bestimmte Formate.

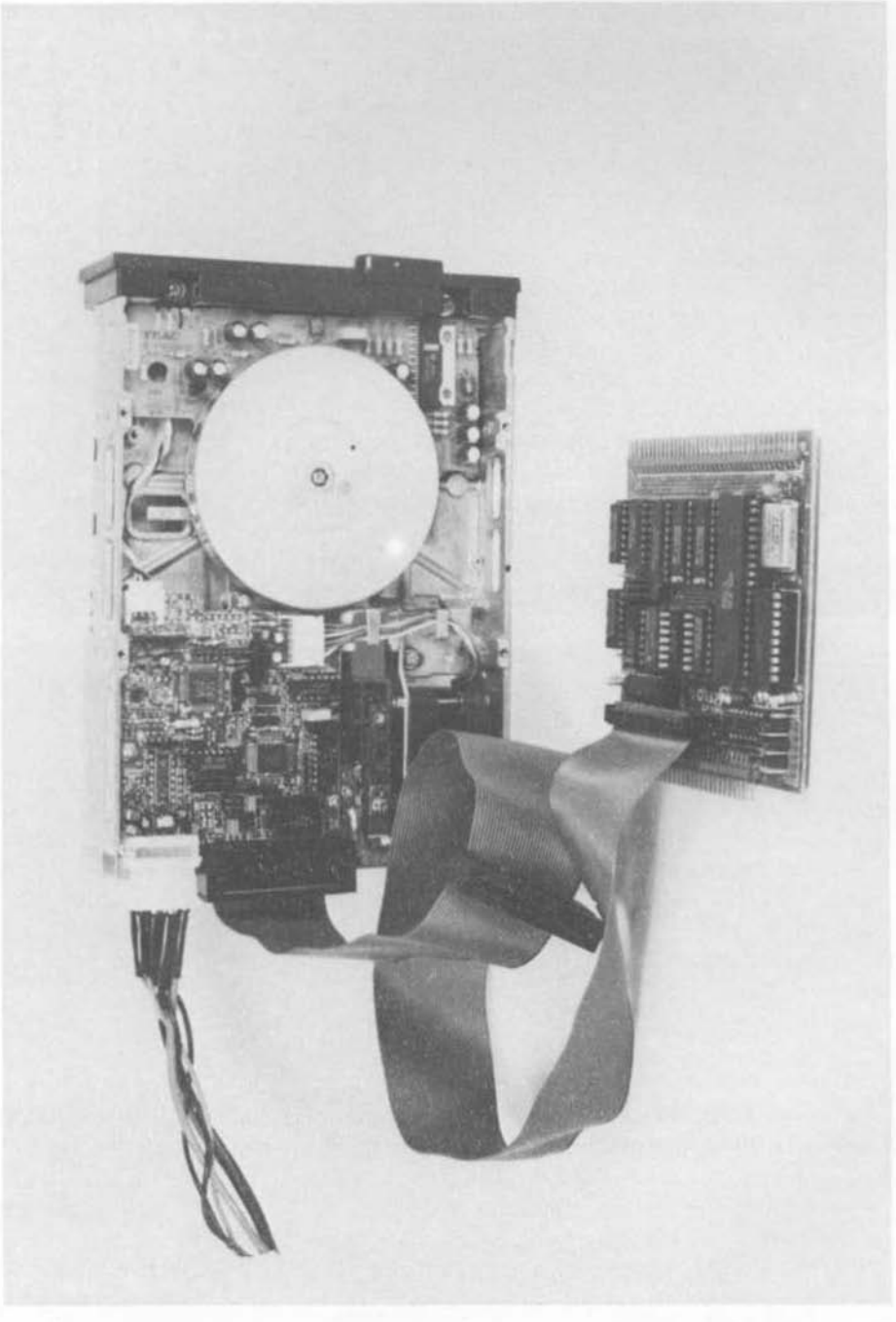


Abb. 6.2.1 Die Verbindung von FLO2 mit dem Laufwerk

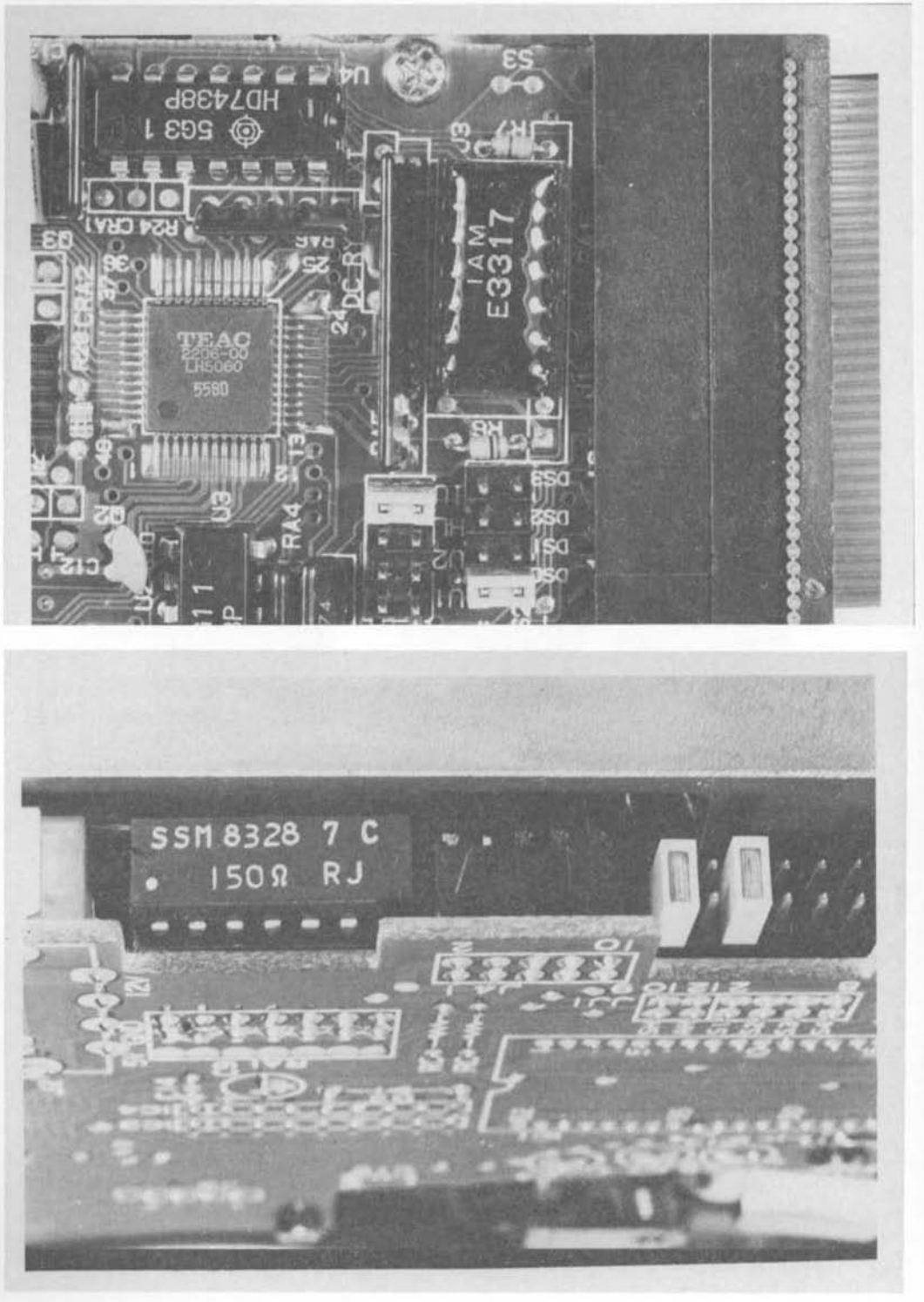


Abb. 6.2.2 Das Widerstandsnetzwerk im Laufwerk

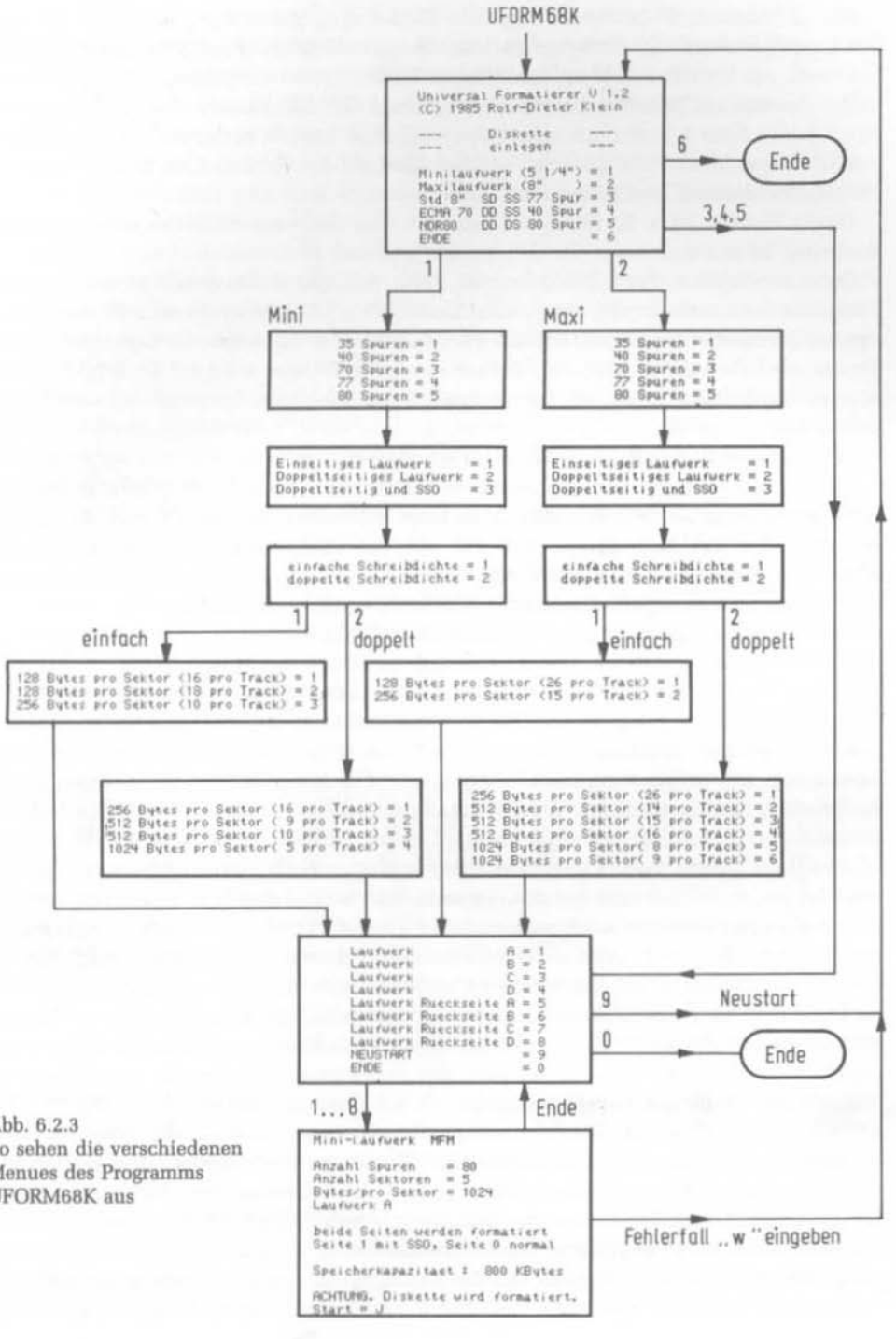


Abb. 6.2.3
So sehen die verschiedenen Menues des Programms UFORM68K aus

Mit „3" werden 8"-Laufwerke im IBM-3740-Format formatiert. Im diesem Format wird normalerweise die Software aus Amerika geliefert. Es erzeugt 77 Spuren mit je 26 Sektoren, die jeweils 128 Byte in einfacher Schreibdichte aufnehmen.

Das Format „ECMA-70" ist ein Norm-Format für Mini-Laufwerke. Es erzeugt 40 Spuren, mit Spur 0 in einfacher und den restlichen Spuren in doppelter Dichte. Dabei wird auf Spur 0 mit 16 Sektoren zu 128 Byte und auf den Spuren 1 bis 39 mit doppelter Dichte, 16 Sektoren zu 256 Byte formatiert.

Dieses Format ist z. B. für den Austausch von Software zwischen verschiedenen Rechnern interessant.

Dann schließlich das NDR80-Format. Dies soll unser Standard-Format für 5¼"-Disketten (wie auch für die enstprechenden 3½" und 3"-Disketten) sein. Es werden 80 Spuren formatiert. Jede Spur enthält 5 Sektoren mit je 1024 Byte in doppelter Dichte. Ferner wird die Vorder- und die Rückseite formatiert. Dabei wird auf der Rückseite das sogenannte Seiten-Auswahl-Bit gesetzt, um dem Disketten-Controller eine zusätzliche Information zu geben.

Mit dem Menue-Punkt „6" wird das Formatieren beendet.

Nun zu den weiteren Menuepunkten. Wählt man „1" oder „2", so gelangt man in ein Menue zur Angabe der Spurzahl. Man kann zwischen 35,40,70,77 und 80 Spuren wählen. Dies sind alle gängigen Werte, die bei Disketten-Laufwerken vorkommen können. Hat man eine Spurzahl ausgewählt, wird gefragt, ob das Laufwerk „einseitig", „doppelseitig" oder „doppelseitig mit SSO" formatiert werden soll.

Die Formatierung der beiden Seiten beim doppelseitigen Laufwerk erfolgt abwechselnd. Es wird zunächst Spur 0 der Seite 0, dann Spur 0 der Seite 1, gefolgt von Spur 1 der Seite 0 usw. bearbeitet. Was bedeutet die Auswahl zwischen Nicht-„SSO" und „SSO"? SSO ist ein Ausgang am Disketten-Controller, er bedeutet Side Select Output. Dieser Ausgang ist normalerweise nicht beschaltet, er hat aber dennoch eine Bedeutung. Zusammen mit seiner Programmierung (ob er ein 1-Signal oder ein 0-Signal führt) ändern sich die Eigenschaften beim Formatieren und auch beim Lesen und Schreiben. Wenn mit „SSO" formatiert wird, so wird auf der Rückseite der Diskette das Seiten-Auswahl-Bit gesetzt. Beim Lesen oder Schreiben wird das Bit immer zusammen mit der Sektor- und Spur-Nummer geprüft. Es muß also beim Lesen und Schreiben mit der Nummer beim Formatieren übereinstimmen. Nun würde man normalerweise dieses Bit immer bei der Rückseite setzen, denn dafür ist es ja da, aber es gibt einige Computer, bei denen das nicht geschieht. Um solche Formate erzeugen zu können, gibt es die Auswahl im Menue. Wichtig ist dann, daß man beim Lesen oder Schreiben dem Controller auch mitteilt, ob das Bit gesetzt ist oder nicht. Dazu gibt es einen speziellen Befehl, mit dem gleichzeitig die Schritt-Geschwindigkeit des Schrittmotors eingestellt wird. Dies werden wir noch später kennenlernen.

Nach dem Menue zur Seitenauswahl wird abgefragt, ob man mit einfacher oder doppelter Schreibdichte formatieren will. Will man im 68008-System 8"-Disketten mit doppelter Schreibdichte formatieren, erscheint die Fehlermeldung CPU zu langsam. Das gleiche gilt auch, wenn man den 68008 mit weniger als 8MHz, oder zu vielen Wait-Zyklen betreibt und ein anderes Format verwendet.

Der 68008 sollte beim Betrieb mit der FLO2 mit genau drei Wait-Zyklen gefahren werden, die Brücke ist also auf der dritten Position von links nach rechts zu stecken.

8" und doppelte Dichte läßt sich nur mit der großen 68000 CPU in der 12 MHz-Version formatieren und bearbeiten. Das kommt daher, daß alle Unterprogramme für den Floppy-Betrieb ohne Interrupttechnik arbeiten und zeitkritisch sind.

Wir wollen das 8"-Format ohnehin höchstens mit einfacher Dichte verwenden, daher stört diese Einschränkung nicht.

Nun verzweigt sich das Menue. Es wird nun ausgewählt, wieviele Bytes pro Sektor und wieviele Sektoren pro Spur formatiert werden sollen. Dann gelangt man in das Laufwerks-Auswahl-Menue.

Zunächst kann man zwischen den Laufwerken A bis D wählen. Dabei ist Laufwerk „A" das Laufwerk, das mit der Brücke „DS0" oder „S1" auf dem Laufwerk, ausgewählt wurde, Laufwerk „B" gehört zu „DS1" oder „S2" usw. Wenn man im Menue ein doppelseitiges Laufwerk ausgewählt hat, so wird automatisch die Rückseite mit formatiert.

In diesem Laufwerks-Auswahl-Menue kann man aber auch Rückseiten direkt ansprechen. „Laufwerk Rückseite A" wählt die Rückseite von Laufwerk „A" zur Formatierung aus. Dabei muß im Menue zuvor ein einseitiges Laufwerk angegeben werden. So ist es möglich, die Rückseiten auch einzeln zu formatieren.

Nach der Laufwerksangabe erscheint ein Kontroll-Menue Hier kann man nochmals alle Einstellungen überprüfen. Drückt man die Taste „J", beginnt die Formatierung. Achtung, alle gegebenenfalls vorhandenen Daten der Diskette werden gelöscht! Für jede formatierte Spur wird das Zeichen „F" auf dem Bildschirm ausgegeben.
Dann wird die Formatierung überprüft. Jetzt wird pro Spur ein „V" auf dem Bildschirm ausgegeben. Waren alle Spuren in Ordnung, gelangt man anschließend wieder in das Laufwerks-Auswahl-Menue zurück.

Das Formatierprogramm reagiert auf verschiedene Fehler, die wir kurz einmal durchgehen wollen.

Abb. 6.2.4 zeigt die Meldung „CPU zu langsam". Das kann passieren, wenn man zuviele Wait-Zyklen auf der CPU-Baugruppe gesteckt hat, oder wenn man einen falschen Quarz verwendet. Außerdem erscheint die Meldung beim 68008, wenn man 8"-Laufwerke mit doppelter Dichte formatieren will.

Oben sind ferner die Spur und der Sektor angegeben, bei dem der Fehler auftrat. Hier ist es Sektor 6, obwohl nur die Sektoren 1 bis 5 formatiert wurden. Den Fehler kann man erst nach der Formatierung einer kompletten Spur erkennen.

Eine andere Fehlermöglichkeit besteht durch einen gesetzten Schreibschutz. Entweder ist der Schreibschutz fälschlicherweise aktiv, oder ein Fehler liegt vor, der dies vortäuscht. So kann auch eine Leitungsunterbrechung im Flachbandkabel den Fehler verursachen.

Einen fatalen Fehler demonstriert die Meldung aus *Abb. 6.2.5*. Sie besagt, daß im Prinzip schon irgendwelche Daten ankommen, aber daß sie fehlerhaft sind. Man sollte hier z. B. die Einstellung der Precompensation auf der FLO2-Baugruppe prüfen (siehe Kapitel 7) und die Daten nachmessen.

Ist es Ihnen geglückt, eine Diskette zu formatieren, so ist ein großes Stück Arbeit geleistet, man kann gratulieren. Jetzt kann eigentlich nichts mehr passieren, denn das Formatier-Programm prüft alle kritischen Werte mit. Schreiben und Lesen wird jetzt automatisch funktionieren.

Bitte formatieren Sie nun ein paar Disketten im NDR80-Format, so daß sie nachher für Versuche bereitliegen. Besitzer von anderen Laufwerken können auch mitmachen. Nur vor der Behandlung des Mikrodos ist dieses entsprechend anzupassen.

Die Routinen zum Lesen und Schreiben

Im Grundprogramm, Version 4.3, befinden sich bereits Unterprogramme für den Floppy-Betrieb. Im Prinzip haben wir nur zwei verschiedene Aufgaben an diese Routinen zu vergeben. Wir wollen entweder einen Sektor auf die Floppy schreiben, oder wir wollen ihn wieder lesen. Neben der Sektornummer wollen wir dabei natürlich die Spur und das Laufwerk angeben können. Dann muß man noch einen Hinweis geben, in welcher Dichte der Sektor geschrieben wird und auf welcher Seite sich der Sektor befindet.

Für dieses Vorhaben gibt es das Unterprogramm FLOPPY, das sowohl lesen als auch schreiben kann. Es erhält verschiedene Parameter beim Aufruf.

Im Register A0 steht die Adresse des Ziels (beim Lesen von der Diskette) oder der Quelle (beim Schreiben zur Diskette) im Speicher.
Im Register D1.W wird ein Befehlscode erwartet. Steht dort der Wert 0, so wird der Inhalt des Registers D3.B zur Einstellung der Laufwerksparameter verwendet. Diese Einstellung legt die Steprate des Laufwerks und gleichzeitig den Test auf das Seiten-Auswahl-Bit fest.

Folgendes wird im Register D3.B erwartet: Die Steprate nimmt den Wert 0 (schnellste Rate) bis 3 (langsamste Rate) an. Die langsamste Rate ist nach dem Einschalten voreingestellt. Die mögliche Einstellung hängt vom Laufwerk ab. Alle modernen Laufwerke können mit 3 ms arbeiten. Wie aus dem Schema der *Abb. 6.2.6* zu erkennen ist, kann man bei Mini-Laufwerken maximal 6 ms verwenden. Möchte man schneller arbeiten, so ist eine Routine zu schreiben, die kurz auf Maxi umschaltet. Wenn man auf der Rückseite mit dem Side-Select-Bit arbeitet, wie es das NDR80-Format tut, muß man den Wert $80 addieren, also das höchste Bit im Steuerbyte setzen. Dann wird bei jedem nachfolgenden Zugriff auf die Rückseite der Floppy automatisch die Stellung des Seiten-Auswahl-Bits mit 1 anstatt mit 0 verglichen. Bei der Bearbeitung der Vorderseite ändert sich nichts dadurch.

Eine 1 im Register D1.W steht für „Sektor lesen", eine 2 für „Sektor schreiben". Dazu werden weitere Register benötigt.

Im Register D2.B steht die Sektornummer. Beim NDR80 Format ein Wert zwischen 1 und 5. Beim 8" Format, einfache Dichte, ein Wert zwischen 1 und 26. Die Länge des Sektors wird vom Programm anhand der Formatierung auf der Diskette automatisch festgestellt. Beim NDR80 Format ist ein Sektor genau 1024 Byte lang, bei 8", einfache Dichte, 128 Byte.

Im Register D3.B wird die Spur übergeben. Der Wert ist bei 80-Spur-Laufwerken zwischen 0 und 79 anzugeben, bei 8"-Laufwerken zwischen 0 und 76 Spuren. Hier fängt man also bei 0 an zu zählen, während man bei der Sektornummer normalerweise bei 1 anfängt.

```
Diskette wird formatiert. Bitte warten.


 Spur = 0
 Sektor = 6

CPU zu langsam fuer gewaehltes Format,
Datenverlust.
Hoehere Taktfrequenz noetig.
Oder Anzahl der Wait-Zyklen zu gross.

weiter = w
```

Abb. 6.2.4
Die CPU ist zu langsam

```
Diskette wird formatiert. Bitte warten.

FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
Diskette wird geprueft. Bitte warten.

UUUUUU
 Spur = 6
 Sektor = 3

Fehler: CRC-Fehler aufgetreten.

weiter = w
```

Abb. 6.2.5
Fehler beim Prüflesen

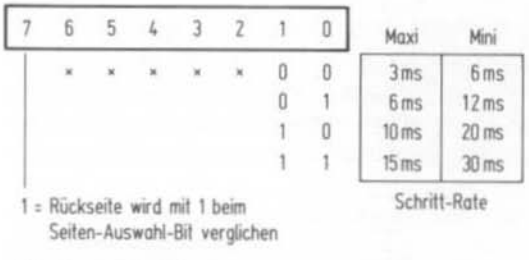


Abb. 6.2.6 Die Belegung von Register D3.B

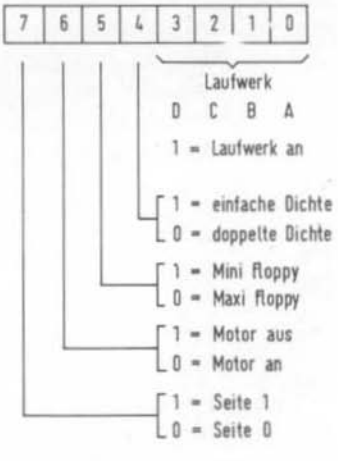


Abb. 6.2.7 Die Belegung des Ports $FFFFFFC4

Im Register D4.B steht der kombinierte Laufwerksdichtcode, der exakt dem Port $FFFFFFC4 des Floppy-Controllers entspricht. *Abb. 6.2.7* zeigt die Belegung der einzelnen Bits.

Das Bit 7 bestimmt, ob auf die Vorder- oder Rückseite geschrieben wird, 0 bedeutet dabei Vorderseite. Mit Bit 6 kann man den Laufwerksmotor abstellen. Dies sollte man durch direktes Schreiben auf den Port $FFFFFFC4 tun und nicht mit einem Floppy-Zugriff. Das Bit 6 wird also normalerweise auf 0 stehen. Bit 5 gibt an, ob ein Mini- (1), oder ein Maxi-Laufwerk verwendet wird. Mit Bit 4 wird die Schreibdichte eingestellt. Bei 0 wird die doppelte (MFM), bei 1 die einfache Schreibdichte verwendet (FM). Damit ergeben sich 4 verschiedene Übertragungsraten. Minilaufwerke mit einfacher Dichte werden mit 125 KBit pro Sekunde bedient, Minilaufwerke mit doppelter Dichte, sowie Maxilaufwerke mit einfacher Dichte werden mit 250 KBit/s und Maxilaufwerke mit doppelter Dichte mit 500 KBit/s bedient. Das ist auch der Grund, warum der 68008 beim letzten Format in der 8-MHz-Version Probleme bekommt. Er hat nur noch 16 µs Zeit, um ein Byte abzuholen, dabei muß er auch noch den Status etc. abfragen. In den Bits 0 bis 3 des Registers D4.B steht der Laufwerkscode. Dabei steht der Wert 1 für das Laufwerk A, 2 für B, 4 für C und 8 für D. Will man mehr als 4 Laufwerke verwenden, so muß man eine binäre Codierung verwenden, über die aber nicht alle Laufwerke verfügen.

Nach Aufruf des Unterprogramms FLOPPY erscheint im Register D0.W ein Fehlercode. Null steht dabei für Fehlerfreiheit, $FFFF für einen Fehler. Im Lesebetrieb lag z.B. ein CRC-Fehler oder ein falsch eingestellter Laufwerkscode vor. Wenn beim Schreiben ein Fehler auftrat, so war entweder der Dichtecode falsch, oder die CPU wurde mit einer falschen Taktrate (kleiner als 8MHz) betrieben.

Bevor wir ein paar Beispiele behandeln, noch ein wichtiges Unterprogramm: GETFLOP. Es stellt die Dichte eines Laufwerks automatisch fest. Wenn man im Register D4 den Laufwerkscode angibt, ohne dabei die Bits 4 bis 7 zu setzen, so werden sie durch GETFLOP automatisch richtig, dem Format der eingelegten Diskette entsprechend, gesetzt. Damit kann man automatische Starts von Programmen auf Diskette durchführen. So wird die Routine vom Grundprogramm dazu verwendet, den Menue-Punkt „Floppy-Start" zu realisieren. Es wird der Sektor 1 auf Spur 0 gelesen, die Daten werden, beginnend bei der Adresse $9800, im Speicher abgelegt. Dabei ist das Format dieses Sektors gleichgültig. Steht danach an der ersten Stelle dieser Daten, also auf der Adresse $9800 der Befehl NOP, so startet das Grundprogramm dieses Programm automatisch.

Wir wollen nun ein Programm schreiben, das ein kleines Schildkrötenprogramm auf der Diskette so ablegt, daß es später durch diese Autostart-Funktion geladen und gestartet werden kann.

Die *Abb. 6.2.8* zeigt das Listing, so wie es im Editor erscheint. Wählen Sie im Menue „TEXTSTART" innerhalb des „Optionen"-Menues die Adresse $9000 neu aus und tippen Sie das Programm dann ein.

Nach der Übersetzung wird es, beginnend bei der Adresse „speichern", gestartet. Die Floppy klickt kurz, das Programm ist abgespeichert. Dabei wird zunächst mit „GETFLOP" das Format erkannt. Das kann 1 s dauern, es geht jedoch schneller, wenn man in D4 gleich den Dichtecode mit einträgt.

Dann wird das Programm ab der Adresse „starte" auf den Sektor 0 der Spur 1 gespeichert. Wieviele Bytes geschrieben werden, hängt von der Formatierung ab. Da das Programm aber kleiner als 128 Byte ist, wird es auf jeden Fall ganz abgespeichert. Anschließend wird noch das Textgebiet auf den Sektor 2 geschrieben, so daß der Quelltext auch gerettet wird. Dabei muß man aufpassen, denn der Text ist fast 1024 Byte lang, und er wird nur dann ganz abgespeichert, wenn man 1 K pro Sektor als Format verwendet (z. B. NDR80).

Jetzt kann man ins Menue „Floppy-Start" gehen und es aufrufen. Die Floppy greift zu, und es muß eine senkrechte Linie erscheinen. Wenn man irgendeine Taste danach drückt, gelangt man wieder ins Grundprogramm zurück. Daher ist im Programm der Aufruf „JSR @CI" vorhanden, denn beim Starten mit „Floppy-Start" würde das Programm sonst sofort wieder ins Grundprogramm zurückkehren, und man bekäme die Linie nie zu sehen. Der Aufruf von „GRAPOFF" ist hier nötig, da sonst keine Rücksetzung der Schildkrötengraphik vorgenommen werden würde.
Und nun ein gewagtes Manöver: nehmen Sie die Diskette aus dem Computer, und schalten Sie ihn aus. Dann schalten Sie den Computer wieder ein, legen die Diskette wieder ein und rufen das Menue „Floppy-Start" auf. Dann muß wieder die Linie erscheinen.
Wie bekommen wir nun die Textquelle wieder zurück? Dazu benötigen wir das kleine Ladeprogramm aus der *Abb. 6.2.9*. Wenn man es durch den Aufruf des Namens „lade" startet, so wird der Programmtext des vorherigen Programms auf Adresse $A000 zurückgeladen. Nun kommt ein toller Trick. Der Textstart wird mit dem Unterprogramm „PUTSTX" automatisch neu festgelegt, und mit „JSR @EDIT" wird der Texteditor aufgerufen. Man bekommt also gleich das alte Programm in den Texteditor. Wenn man den Texteditor wieder verläßt, das geht ganz normal mit CTRL-„K" und „X", so wird durch ein weiteres „PUTSTX" der Textstart wieder zurückgesetzt. Ruft man nun den Editor vom Menue aus auf, erscheint der Text des Lade-Programms.

Achtung, hier noch ein Hinweis zur Benutzung der Disketten-Rückseite: Beim NDR80-Format wird das Seiten-Auswahl-Bit (SSO) auf der Rückseite gesetzt. Im Grundprogramm muß man dies vor Benutzung eines Floppy-Befehls angeben. *Abb. 6.2.10* zeigt ein Beispiel. Dazu verwendet man den Code 0 im Register D1. Dann kann man im Register D3 den kombinierten Code - Steprate und SSO -angeben. Ist Bit 7 gesetzt, so wird künftig bei jedem Zugriff auf die Rückseite das SSO-Bit mit dem Wert 1 verglichen, so wie es das NDR80-Format will. Im Beispiel-Programm wird daher der Wert $80 in D3 geladen. Gleichzeitig wird die langsamste Steprate eingestellt, da die Bits 0 und 1 null sind. Will man nun auf die Rückseite zugreifen, so muß man für den nächsten Aufruf von „FLOPPY" im Register D4 das Bit 7 auf 1 setzen. Im Beispielprogramm geschieht dies durch den „OR #$80,D4"-Befehl. Wenn man dieses Programm ausführt, so wird der erste Sektor auf Spur 0 der Rückseite gelesen. Dort steht die Formatier-Information „E5" als Datenwert. Man kann sich den Inhalt nach dem Ablauf des Programms ansehen, indem man im Menue „Speicher ansehen" die Adresse $A000 angibt.
Zum Lesen der Vorderseite ist einfach der „OR #$80,D4"-Befehl aus dem Programm zu streichen. Es wird dann der erste Sektor der Vorderseite geladen, der unser Schildkröten-Programm enthält.

```
start:
 nop             * Erkennung fuer Auto-Load
 move #100,d0    * hier beliebiges Programm
 jsr @schreite   * eingeben
 jsr @ci         * warten bis Taste da
 jsr @grapoff    * sonst bleibt Schildkroete
 rts             * Programmende

speichern:       * Ablegen des Sektors
 move #1,d4      * Laufwerk A
 jsr @getflop    * Dichte bestimmen
 lea start,a0    * Programm-Start
 move #2,d1      * Befehl: Schreiben
 move #1,d2      * Sektor 1
 move #0,d3      * Spur 0
 jsr @floppy     * in D4 steht Dichtecode
 lea $9000,a0    * Text auch speichern
 move #2,d1      * schreiben,einfach auf
 move #2,d2      * Sektor 2
 move #0,d3      * Track 0
 jsr @floppy     * alles ohne Fehlercheck
 rts

 end
```

```
Textstart=009000 Fenster=009032 Ter=00903A      aner CTRL-J=Hilfe
```

Abb. 6.2.8
Das Programm speichert sich selbst ab und kann durch „Floppy-Start" aktiviert werden

```
lade:
 move #$1,d4     * Laufwerk A
 jsr @getflop    * Dichte bestimmen
 lea $a000,a0    * Text dorthin
 move #1,d1      * Lesen
 move #2,d2      * Sektor 2
 move #0,d3      * Spur 0
 jsr @floppy     * und ausfuehren
 move.l #$a000,d0 * und Textstart setzen
 jsr @putstx     * fuer Editor
 jsr @edit       * und gleich aufrufen
 move.l #$9000,d0 * wieder alten Text
 jsr @putstx     * anwaehlen
 rts             * Ende
```

Abb. 6.2.9
Das Programm lädt die Textquelle von der Diskette in den Speicher ab $A000 und ruft den Editor auf

```
lade:
 move #$1,d4     * Laufwerk A
 jsr @getflop    * Dichte bestimmen
 move #0,d1      * Steprate u. SSO
 move #$80,d3    * Rueckseite mit SSO
 jsr @floppy     * einstellen
 lea $a000,a0    * Text dorthin
 move #1,d1      * Lesen
 move #1,d2      * Sektor 1
 move #0,d3      * Spur 0
 or #$80,d4      * Rueckseite lesen
 jsr @floppy     * und ausfuehren
 rts             * Ende
```

Abb. 6.2.10
Die Benutzung der Disketten-Rückseite

Das gleiche gilt beim Schreiben. Wenn man vergißt, das SSO-Bit zu setzen, so erhält man einen Lese-Fehler. Der Sektor wird auch nicht geladen. Also beim Verwenden der Rückseite aufpassen. Im nächsten Abschnitt werden Sie ein Programm kennenlernen, das all dies automatisch berücksichtigt und die 80-Spur-Laufwerke auf Vorder- und Rückseite richtig bedient.

6.3 Das Mikrodos

Wie arbeitet ein Betriebssystem?

Man könnte nun mit dem soeben beschriebenen Unterprogramm Programme und Texte auf einer Diskette abspeichern. Dazu müßte man sich auf einer Karteikarte den Namen des Programms und die Sektoren sowie die Spuren notieren, wo das Programm zu finden ist. Einfacher geht das, wenn man ein Programm verwendet, das diese Verwaltungsarbeit automatisch ausführt und selbst eine Art Karteikartensystem führt. Ein solches Karteikartensystem nennt man auch Inhaltsverzeichnis oder engl. directory. Man erhält ein DOS, ein disk operating system, oder ein Diskettenbetriebssystem. Ein einfaches DOS, das man auch praktisch verwenden kann, um Programme auf einer Diskette zu speichern und sie wieder zu laden, wollen wir uns nun einmal ansehen. *Abb. 6.3.1* zeigt das Listing des Programms.

Der Programmtext wird ab Adresse $A000 eingegeben, das ist sehr wichtig. Der Maschinencode selbst kommt ab der Adresse $9800 zu liegen. Das ist die Adresse, auf der es später auch vom Grundprogramm beim Floppy-Start geladen wird.

Achtung, noch ein Hinweis: Kontrollieren Sie das abgetippte Programm sehr sorgfältig, und vergleichen Sie es mit dem abgedruckten Listing, denn ein Fehler, und alles war umsonst. Vielleicht ist es auch ganz gut, wenn Sie vor dem Start das Programm noch auf die Kassette abspeichern, dabei wird $A000 als Startadresse verwendet.
Das Programm wird nun mit dem Assembler übersetzt und anschließend gestartet. Ein kleines Menue wird sichtbar.

Es ist jetzt sehr wichtig, daß zuerst das Menue „3" aufgerufen wird, also die Funktion „Speich." = Speichern. Eine im NDR80-Format formatierte Diskette muß nun im Laufwerk „A" vorhanden sein.

Es erscheint nun *Abb. 6.3.2*. In das Fenster trägt man den Namen des abzuspeichernden Programms ein, hier „dos4". Damit werden die Quelle des Mikrodos und der Objektcode ab Adresse $9800 auf die Diskette gespeichert. Das Mikrodos ist so geschrieben, daß es sich selbst auf der Diskette abspeichert. Jetzt ist das Mikrodos betriebsbereit.

Das DOS arbeitet mit dem NDR80-Format, um eine maximale Kapazität zu erreichen. Wer andere Laufwerke besitzt, muß das Programm entsprechend ändern. Dabei sollte aber immer ein Format mit 1024 Byte pro Sektor verwendet werden, also z.B. nicht das ECMA70-Format.

Nun kommt der große Test. Nehmen Sie die Diskette aus dem Computer und schalten Sie ihn aus. Dann schalten Sie ihn wieder ein und legen die Diskette mit dem Mikrodos in das Laufwerk „A". Rufen Sie „4=Floppy start" auf. Es muß sich das kleine Menue melden. Dort geben Sie jetzt „1" ein. Es muß dann *Abb. 6.3.3* erscheinen. Drücken Sie

```
Rolf-D.Klein 68000/08 Assembler 4.3 (C) 1984, Seite 1
0E9000
009C00                  ****************************************
009C00                  * MIKRODOS, (C) 1985 ROLF-DIETER KLEIN  *
009C00                  * REV 4.0   8504017                     *
009C00                  ****************************************
009C00
= 00000000              GRUNDPROG EQU $0   * HIER BEI VERSCHOBENER VERSION AENDERN
009C00
= 00009000              HILFSSP EQU GRUNDPROG+$9000
= 00009800              AUTOLOAD EQU GRUNDPROG+$9800
= 0000A000              TEXTANF EQU GRUNDPROG+$A000   * TEXT HIER ABLEGEN
009C00
= 00000000              SDIR EQU 0  * KENNUNG
= 00000002              SNAM EQU 2  * NAME
= 0000000A              SFUL EQU 10 * DUMMY
= 0000000C              SSEK EQU 12 * ERSTE SEKTOR
= 0000000E              STRK EQU 14 * TRACK
009C00
= 00009800              START EQU AUTOLOAD
009C00
009C00                   OFFSET AUTOLOAD
000000                   ORG 0
000000
000000                  LOOP:           * SPAETER AUF $9800 BIS $9BFF
000000  4E71             NOP             * KENNUNG FUER AUTOLOAD
000002  49FA FFFC       LEA LOOP(PC),A4  * ALS INDEX FUER RELTIVEN ZUGRIFF
000006  4241             CLR D1          *
000008  363C 0083       MOVE #$83,D3    * BEFEHL STEPRATE UND SSO
00000C  3E3C 004B       MOVE #75,D7
000010  4E41             TRAP #1         * EINSTELLEN AN DER FLOPPY
000012  41FA 0362       LEA MENUE(PC),A0
000016  6100 0306       BSR PRINT       * MENUE AUSGEBEN
00001A  6100 02BE       BSR CI
00001E  0C00 0031       CMP.B #'1',D0   * INHALT
000022  6700 01F4       BEQ INHALT
000026  0C00 0032       CMP.B #'2',D0   * LADEN
00002A  6700 00F8       BEQ LADEN
00002E  0C00 0033       CMP.B #'3',D0
000032  6600 0302       BNE RETEX       * ENDE, SONST SPEICHERN
000036  6100 01A6       BSR GETNAM
00003A  4A04             TST.B D4
00003C  6700 02F8       BEQ RETEX      * KEIN NAME DA
000040
000040                  SPEICHERN:      * SPEICHERN
000040                                  * EINTRAGEN UND SPEICHERN
000040                  * ERST DATEN
000040  342C 034C        MOVE SEKTOR(A4),D2
000044  362C 034E        MOVE TRACK(A4),D3
000048  41F9 0000A000    LEA TEXTANF,A0          * START
00004E  383C 0021        MOVE #$21,D4
000052                  SPELP:
000052  7202             MOVEQ #2,D1             * SCHREIBEN
000054  6100 02A0        BSR FLOPPY
000058  6500 02E4        BCS ERROR
```

zu Abb. 6.3.1

```
Rolf-D.Klein 68000/08 Assembler 4.3 (C) 1984, Seite 2

00005C  5242             ADDQ #1,D2              * NEUER SEKTOR
00005E  0C42 0006        CMP #6,D2               * 1..5
000062  660C             BNE.S SPE3CON
000064  7401             MOVEQ #1,D2
000066  5243             ADDQ #1,D3
000068  0C43 00A0        CMP #160,D3             * 0..159, DA DOPPELSEITIG
00006C  6700 02D0        BEQ ERROR               * IMMER WEITER SO.
000070                   SPE3CON:
000070  3A3C 03FF        MOVE #1024-1,D5         * ENDE SUCHEN
000074                   SPE2LP:
000074  4A18             TST.B (A0)+
000076  6700 000A        BEQ OKSENDE
00007A  51CD FFF8        DBRA D5,SPE2LP
00007E  6000 FFD2        BRA SPELP
000082                   OKSENDE:
000082  3942 0350        MOVE D2,SEKNEU(A4)
000086  3943 0352        MOVE D3,TRKNEU(A4)
00008A                   *
00008A  41F9 00009000    LEA HILFSSP,A0           * DIREKTORY ABLEGEN
000090  7201             MOVEQ #1,D1
000092  342C 0354        MOVE DIRSEK(A4),D2
000096  0C42 0006        CMP #6,D2               * DIREKTORY VOLL
00009A  6700 029C        BEQ FULERR
00009E  4243             CLR D3                  * DIREKTOR WAECHST JEDESMAL
0000A0  383C 0021        MOVE #$21,D4
0000A4  6100 0250        BSR FLOPPY              * DIR HOLEN, A0 BLEIBT
0000A8  6500 0294        BCS ERROR
0000AC  3A2C 0356        MOVE DIRBYTE(A4),D5
0000B0  4270 5000        CLR.W SDIR(A0,D5.W)
0000B4  21AC 03A2 5002   MOVE.L BUFFER(A4),SNAM(A0,D5.W)
0000BA  21AC 03A6 5006   MOVE.L BUFFER+4(A4),SNAM+4(A0,D5.W)
0000C0  4270 500A        CLR.W SFUL(A0,D5.W)
0000C4  31AC 034C 500C   MOVE.W SEKTOR(A4),SSEK(A0,D5.W)
0000CA  31AC 034E 500E   MOVE.W TRACK(A4),STRK(A0,D5.W)
0000D0  323C 0002        MOVE #2,D1
0000D4  342C 0354        MOVE DIRSEK(A4),D2
0000D8  363C 0000        MOVE #0,D3
0000DC  383C 0021        MOVE #$21,D4
0000E0  6100 0214        BSR FLOPPY
0000E4  6500 0258        BCS ERROR
0000E8                   *                                          zu Abb. 6.3.1
0000E8  396C 0350 034C   MOVE SEKNEU(A4),SEKTOR(A4)
0000EE  396C 0352 034E   MOVE TRKNEU(A4),TRACK(A4)
0000F4  066C 0010 0356   ADD #16,DIRBYTE(A4)
0000FA  0C6C 0400 0356   CMP #1024,DIRBYTE(A4)
000100  6608             BNE.S SPE1
000102  426C 0356        CLR DIRBYTE(A4)
000106  526C 0354        ADDQ #1,DIRSEK(A4) * NEUER SEKTOR
00010A                   SPE1:
00010A  41F9 00009800    LEA AUTOLOAD,A0    * PROGRAMM NEU SPEICHERN
000110  7202             MOVEQ #2,D1        * DADURCH WERDEN ALLE VARIABLE
000112  7401             MOVEQ #1,D2        * GERETTET
000114  4243             CLR D3
```

```
Rolf-D.Klein 68000/08 Assembler 4.3 (C) 1984, Seite 3
000116  383C 0021          MOVE #$21,D4
00011A  6100 01DA          BSR FLOPPY
00011E  6500 021E          BCS ERROR
000122  4E75               RTS
000124
000124
000124                     LADEN:                  * SUCHEN NAMEN, DANN TRK,SEK LADEN
000124  6100 00B8          BSR GETNAM
000128  4A04               TST.B D4
00012A  6700 020A          BEQ RETEX               * KEIN NAME DA
00012E  7A03               MOVEQ #4-1,D5           * ANZAHL
000130  383C 0021          MOVE #$21,D4            * DD
000134  7402               MOVEQ #2,D2             * SEKTOR 2
000136  4243               CLR D3                  * TRACK 0
000138                     LAD1:
000138  41F9 0000A000      LEA TEXTANF,A0          * ZIELADRESSE
00013E  7201               MOVEQ #1,D1             * LESEN
000140  6100 01B4          BSR FLOPPY              * SPEZ.
000144  6500 01F8          BCS ERROR
000148  41F9 0000A000      LEA TEXTANF,A0          * QUELLE
00014E  3C3C 003F          MOVE #64-1,D6           * EINTRAEGE
000152                     LAD2:
000152  4A28 0000          TST.B 0(A0)             * GUELTIG ?
000156  6616               BNE.S LAD3A
000158  2028 0002          MOVE.L SNAM(A0),D0
00015C  B0AC 03A2          CMP.L BUFFER(A4),D0
000160  660C               BNE.S LAD3A
000162  2028 0006          MOVE.L SNAM+4(A0),D0
000166  B0AC 03A6          CMP.L BUFFER+4(A4),D0
00016A  6700 0020          BEQ FOUND               * NAME GEFUNDEN
00016E                     LAD3A:
00016E  D1FC 00000010      ADDA.L #16,A0           * NAECHSTER EINTRAG
000174  51CE FFDC          DBRA D6,LAD2
000178  5242               ADDQ #1,D2              * NEUER SEKTOR MAX 2..5
00017A  51CD FFBC          DBRA D5,LAD1
00017E  41FA 01D8          LEA NONAM(PC),A0
000182  6100 01A0          BSR ERRPRT              * NICHT GEFUNDEN
000186  6100 0152          BSR CI
00018A  4E75               RTS
00018C
00018C                     FOUND:                  * DATEI GEFUNDEN
00018C  3428 000C          MOVE SSEK(A0),D2
000190  3628 000E          MOVE STRK(A0),D3        * START
000194  383C 0021          MOVE #$21,D4            * DICHTE
000198  41F9 0000A000      LEA TEXTANF,A0          * START DORT
00019E                     FNDLP:
00019E  7201               MOVEQ #1,D1
0001A0  6100 0154          BSR FLOPPY
0001A4  6500 0198          BCS ERROR
0001A8  3A3C 03FF          MOVE #1024-1,D5         * ENDE SUCHEN
0001AC                     FND2LP:
0001AC  4A18               TST.B (A0)+
0001AE  6700 0020          BEQ OKENDE
0001B2  51CD FFF8          DBRA D5,FND2LP
```

zu Abb. 6.3.1

```
Rolf-D.Klein 68000/08 Assembler 4.3 (C) 1984, Seite 4

0001B6  5242             ADDQ #1,D2              * NEUER SEKTOR
0001B8  0C42 0006        CMP #6,D2               * 1..5
0001BC  6600 FFE0        BNE FNDLP
0001C0  7401             MOVEQ #1,D2
0001C2  5243             ADDQ #1,D3
0001C4  0C43 00A0        CMP #160,D3             * 0..159
0001C8  6600 FFD4        BNE FNDLP               * IMMER WEITER SO.
0001CC  6000 0170        BRA ERROR               * FATAL ERROR
0001D0                   OKENDE:
0001D0  203C 0000A000    MOVE.L #TEXTANF,D0      * UND TEXTSTART SETZEN
0001D6  3E3C 0043        MOVE #67,D7
0001DA  4E41             TRAP #1                 * PUTSTX, NEUER TEXTSTART
0001DC                   FINAL:
0001DC  4E75             RTS
0001DE
0001DE                   GETNAM:                 * NAME ANLEGEN
0001DE  41FA 01AE        LEA NAME(PC),A0
0001E2  303C 0022        MOVE #$22,D0
0001E6  323C 00FA        MOVE #250,D1
0001EA  343C 0064        MOVE #100,D2
0001EE  3E3C 000A        MOVE #10,D7
0001F2  4E41             TRAP #1                 * NAME:
0001F4  42AC 03A2        CLR.L BUFFER(A4)
0001F8  42AC 03A6        CLR.L BUFFER+4(A4)
0001FC  41EC 03A2        LEA BUFFER(A4),A0 * ZIEL
000200  303C 0022        MOVE #$22,D0
000204  323C 0154        MOVE #340,D1
000208  343C 0064        MOVE #100,D2
00020C  363C 0008        MOVE #8,D3              * MAX 8 ZEICHEN
000210  3E3C 000B        MOVE #11,D7
000214  4E41             TRAP #1                 * READ
000216  4E75             RTS
000218
000218
000218                   INHALT:                 * INHALTSVERZEICHNIS AUSGEBEN
000218  3E3C 0014        MOVE #20,D7
00021C  4E41             TRAP #1                 * CLRSCREEN
00021E  7A03             MOVEQ #4-1,D5           * ANZAHL
000220  383C 0021        MOVE #$21,D4            * DD
000224  7402             MOVEQ #2,D2             * SEKTOR 2
000226  4243             CLR D3                  * TRACK 0
000228                   INH1:
000228  41F9 00009000    LEA HILFSSP,A0          * ZIELADRESSE
00022E  7201             MOVEQ #1,D1             * LESEN
000230  6100 00C4        BSR FLOPPY              * SPEZ.
000234  6500 010B        BCS ERROR
000238  41F9 00009000    LEA HILFSSP,A0          * QUELLE
00023E  3C3C 003F        MOVE #64-1,D6           * EINTRAEGE
000242                   INH2:
000242  4A28 0000        TST.B 0(A0)             * GUELTIG ?
000246  6622             BNE.S INH3A
000248  48E7 0280        MOVEM.L D6/A0,-(A7)
00024C  3C3C 0007        MOVE #8-1,D6
000250  D1FC 00000002    ADDA.L #SNAM,A0
```

zu Abb. 6.3.1

```
Rolf-D.Klein 68000/08 Assembler 4.3 (C) 1984, Seite 5

000256                  INH3:
000256  1018            MOVE.B (A0)+,D0
000258  6708            BEQ.S INH3B             * 0= ENDE
00025A  6100 0092       BSR CO2
00025E  51CE FFF6       DBRA D6,INH3
000262                  INH3B:
000262  6100 007E       BSR CRLF
000266  4CDF 0140       MOVEM.L (A7)+,D6/A0
00026A                  INH3A:
00026A  D1FC 00000010   ADDA.L #16,A0           * NAECHSTER EINTRAG
000270  51CE FFD0       DBRA D6,INH2
000274  5242            ADDQ #1,D2              * NEUER SEKTOR MAX 2..5
000276  51CD FFB0       DBRA D5,INH1
00027A  6100 0066       BSR CRLF                * REST INFO AUSGEBEN
00027E  6100 0062       BSR CRLF
000282  6100 005E       BSR CRLF
000286  4240            CLR D0
000288  4241            CLR D1
00028A  3E3C 001B       MOVE #27,D7             * NEWPAGE
00028E  4E41            TRAP #1
000290  41FA 0102       LEA FREI(PC),A0
000294  6100 0084       BSR PRINT2
000298  41EC 03A2       LEA BUFFER(A4),A0
00029C  302C 034E       MOVE TRACK(A4),D0
0002A0  C0FC 0005       MULU #5,D0
0002A4  D06C 034C       ADD SEKTOR(A4),D0
0002A8  0640 FCE0       ADD #-800,D0            * XK FREI
0002AC  4440            NEG D0
0002AE  3E3C 002E       MOVE #46,D7
0002B2  4E41            TRAP #1                 * PRINT4D
0002B4  41EC 03A2       LEA BUFFER(A4),A0
0002B8  6100 0058       BSR PRINT3
0002BC  3E3C 0052       MOVE #82,D7
0002C0  4E41            TRAP #1                 * CURSORMODE AUS. CUROFF
0002C2  4240            CLR D0
0002C4  4241            CLR D1
0002C6  3E3C 0022       MOVE #34,D7
0002CA  4E41            TRAP #1                 * SETFLIP AUF 0
0002CC                  WARTE:
0002CC  6100 000C       BSR CI                  * DANN WARTEN, BIS TASTE
0002D0  0C00 000D       CMP.B #$D,D0            * CR GEDRUECKT WIRD
0002D4  66F6            BNE.S WARTE
0002D6  6000 FD28       BRA LOOP               * DANN NOCHMALS FRAGEN
0002DA
0002DA                  CI:             * ZEICHEN EINLESEN
0002DA  3E3C 000C       MOVE #12,D7
0002DE  4E41            TRAP #1         * CI, AUSWAHL
0002E0  4E75            RTS
0002E2
0002E2                  CRLF:           * ZEILENVORSCHUB GEBEN
0002E2  103C 000D       MOVE.B #$D,D0
0002E6  6100 0006       BSR CO2
0002EA  103C 000A       MOVE.B #$A,D0
0002EE                  CO2:
```

zu Abb. 6.3.1

```
Rolf-D.Klein 68000/08 Assembler 4.3 (C) 1984, Seite 6

0002EE  3E3C 0021        MOVE #33,D7
0002F2  4E41             TRAP #1  * CO2
0002F4  4E75             RTS
0002F6
0002F6
0002F6                   FLOPPY:          * MIT TRACKUMRECHNUNG
0002F6  48E7 7E00        MOVEM.L D1-D6,-(A7)
0002FA  E25B             ROR #1,D3
0002FC  6404             BCC.S FLO1
0002FE  08C4 0007        BSET #7,D4      * RUECKSEITE
000302                   FLO1:
000302  0203 007F        AND.B #$7F,D3
000306  3E3C 004B        MOVE #75,D7
00030A  4E41             TRAP #1         * FLOPPY
00030C  4CDF 007E        MOVEM.L (A7)+,D1-D6
000310  4E75             RTS
000312
000312                   PRINT3:
000312  323C 0110        MOVE #200+72,D1
000316  4242             CLR D2
000318  6012             BRA.S PRTA
00031A                   PRINT2:
00031A  4242             CLR D2
00031C  600A             BRA.S PRT
00031E                   PRINT:
00031E  343C 00C8        MOVE #200,D2
000322  6004             BRA.S PRT
000324                   ERRPRT:
000324  343C 0014        MOVE #20,D2
000328                   PRT:
000328  323C 00C8        MOVE #200,D1
00032C                   PRTA:
00032C  103C 0022        MOVE.B #$22,D0
000330  3E3C 000A        MOVE #10,D7
000334  4E41             TRAP #1         * WRITE
000336                   RETEX:
000336  4E75             RTS
000338
000338                   FULERR:
000338  41FA 0033        LEA DIRFUL(PC),A0
00033C  6004             BRA.S ERR1
00033E                   ERROR:
00033E  41FA 0021        LEA DOSERR(PC),A0
000342                   ERR1:                                    zu Abb. 6.3.1
000342  6100 FFE0        BSR ERRPRT
000346  6100 FF92        BSR CI
00034A  4E75             RTS
00034C
00034C                   * VARIABLE
00034C  0004             SEKTOR: DC.W 4  * FREIER SEKTOR
00034E  0005             TRACK: DC.W 5   * CP/M KOMP.
000350  0004             SEKNEU: DC.W 4
000352  0005             TRKNEU: DC.W 5
```

```
000354  0002            DIRSEK: DC.W 2  * DIREKTORY
000356  0000            DIRBYTE: DC.W 0 * 0,16,...
000358                  * TEXTE
000358  6E696368742064  NONAM: DC.B 'nicht da',0
00035F  6100
000361  4469736B204665  DOSERR: DC.B 'Disk Fehler',0
000368  686C657200
00036D  44697220766F6C  DIRFUL:  DC.B 'Dir voll',0
000374  6C00
000376  4469723D31204C  MENUE: DC.B 'Dir=1 Laden=2 Speich.=3',0
00037D  6164656E3D3220
000384  5370656963682E
00038B  3D3300
00038E  4E616D653A00    NAME: DC.B 'Name:',0
000394  467265693A2020  FREI: DC.B 'Frei:      K',0
00039B  202020204B00
0003A1  00               DS 0  * EVEN
0003A2
0003A2                  BUFFER: DS.B 12          * AUF EVEN, WEGEN .L
0003AE
0003AE                   END

008EAA  Ende-Symboltabelle
```

Abb. 6.3.1 Das Mikrodos – ein kleines Diskettenbetriebssystem

---

```
Dir=1 Laden=2 Speich.=3
```

Abb. 6.3.2
Speichern des Mikrodos

```
Name: dos4
```

```
dos4
```

Abb. 6.3.3
Das Disketten-Inhaltsverzeichnis

```
Frei: 763  K
```

die Taste „CR", das kleine Menue wird wieder eingeblendet. Nun geben Sie „2" ein, denn wir wollen den ursprünglichen Quellentext von unserem Mikrodos laden. Geben Sie als Namen „dos4" an. Der Name muß dabei genauso eingetippt werden, wie ursprünglich beim Speichern, keine Leerzeichen hinter dem Namen und keine Großbuchstaben. Mikrodos ist da sehr empfindlich. Wenn Sie dann „CR" drücken, muß ein Zugriff auf die Diskette erfolgen, anschließend können Sie wieder in das Grundmenue zurück. Dort gehen Sie in den Editor, und das Mikrodos-Programm muß vollständig vorhanden sein. Unser DOS funktioniert.

Sie können nun bis zu 256 Dateien auf der Diskette speichern, die zusammen nicht ganz 800 KByte einnehmen können.

Wenn Sie sehr viele Dateien gespeichert haben, beginnt der Bildschirm bei der Anzeige des Inhaltsverzeichnisses zu rollen (scrolling), sie können die Ausgabe dann durch Eingabe der Tasten CTRL-„S" anhalten und mit CTRL-„Q" wieder fortfahren lassen. Wenn Sie eine Datei laden wollen, die nicht vorhanden ist, erscheint *Abb. 6.3.4* auf dem Bildschirm, „nicht da".

Dateien können nicht wieder gelöscht werden. Das kommt daher, daß unser Mikrodos noch sehr primitiv für ein Disketten-Betriebssystem ist. Als Behelf kann man sich ja zunächst eine Arbeitsdiskette anfertigen und dort alle Entwicklungsphasen eines Programms abspeichern. Funktioniert das Programm dann zufriedenstellend, so wechselt man die Diskette und verwendet eine Archivierungs-Diskette. Die Arbeitsdiskette kann man dann irgendwann einmal neu formatieren und wieder neu verwenden.

Das DOS arbeitet wie folgt. Auf den Sektoren 2 bis 5 der Spur 0 ist das Inhaltsverzeichnis untergebracht, das auch directory genannt wird. Jeder Sektor kann dabei 64 Dateinamen aufnehmen. *Abb. 6.3.5* zeigt den Aufbau eines Directory-Blocks. Das erste Wort dient der Erkennung, ob der Eintrag vorhanden ist oder nicht. Nach dem Formatieren einer Diskette steht dort der Wert $E5E5. Wenn eine Datei eingetragen wird, so wird der Wert 0 eingeschrieben. Dann folgt der Dateiname, der maximal aus 8 Zeichen bestehen kann. Die Bytes 10 und 11 sind mit 0 belegt und können für Erweiterungen verwendet werden. Dann folgt der Startsektor des Eintrags und dahinter die Startspur der Datei. Eine Datei soll aufhören, wenn der Wert 0 in einem Sektor vorkommt, so braucht man sich hier den Endesektor nicht getrennt zu merken. Damit lassen sich aber nur Textdateien abspeichern. Die Größen Sektor und Spur sind normalerweise nur 1 Byte lang, was aber nicht weiter stört.

Unser DOS hat drei verschiedene Aufgaben. Zunächst zum Inhaltverzeichnis. Zur Ausgabe des Inhaltsverzeichnisses werden alle Directory-Sektoren eingelesen; handelt es sich um einen gültigen Eintrag, wird der jeweilige Namenseintrag ausgeben. Am Schluß des Programms wird noch der verbleibende Platz ausgegeben. Dazu wird zunächst der durch voll beschriebene Spuren belegte Platz errechnet, indem der aktuelle Sektor mit fünf multipliziert wird. Zu diesem Wert wird dann der Wert des Sektors addiert, somit erhält man die belegte Kapazität in KByte. Jetzt muß man diesen Wert nur noch von der Gesamtkapazität, also von 800 KByte subtrahieren, und das Ergebnis kann als freier Platz angezeigt werden.

Der Speicherbereich von $9000 bis $93FF wird als Hilfsspeicher verwendet, um z.B. die Directory-Sektoren darin unterzubringen. Der Bereich $9800 bis $9BFF ist für das

Mikrodos reserviert, denn dorthin wird der Maschinencode des Programms geladen. Ab Adresse ‚A000 steht dann der Speicher für Texte frei zur Verfügung.

Beim Laden einer Datei wird zunächst das Inhaltsverzeichnis nach diesem Namen durchsucht. Dabei muß der Name Zeichen für Zeichen übereinstimmen. Leerzeichen hinter dem Namen werden auch mit verglichen, so kann man Geheimdateien aufzeichnen. Auch Groß- und Kleinbuchstaben werden unterschieden. Ist ein Name gefunden, so werden die Daten, ab der im Directory stehenden Spur- und Sektorposition, geladen und ab Adresse $A000 im Speicher abgelegt.

Jetzt zu der kompliziertesten Funktion, zum Speichern. Zum einen muß der neue Name im Inhaltsverzeichnis in den nächsten freien Platz eingetragen werden. Dazu gibt es im Programm den Merker „DIRSEK" und „DIRBYTE". Damit für diese Arbeit genügend Speicherbereich im Textgebiet zur Verfügung steht, wird dieses zuvor ab der Adresse $A000 abgespeichert. Das Floppy-Programm beginnt ja bei der Adresse $9800, der Platz zwischen $9000 und $9800 ist sehr knapp für Texte, daher muß man mit „TEXTSTART=$A000" arbeiten.

Der erste freie Sektor steht in der Variablen „SEKTOR" und die erste freie Spur in der Variablen „TRACK". Es werden nun soviele Sektoren abgespeichert, bis der Wert 0 im Sektor aufgetreten ist. Dann wird der neue Wert des Sektors in „NEUSEK" und die neue Spur in „NEUTRK" abgespeichert. Jetzt wird das Inhaltsverzeichnis geladen, und zwar bei dem Sektor „DIRSEK", dem letzten belegten Sektor. „DIRBYTE" gibt den aktuellen Direktorybuffer im Sektor an. Dorthin wird der Name des Programms, und der Inhalt der Variablen „SEKTOR" und „TRACK" gespeichert. Danach wird der Directory-Sektor auf die Diskette zurückgespeichert. Abschließend wird der nächste Directory-Block berechnet. Der neue Eintrag belegt einen Platz von 16 Byte. Also wird zunächst „DIRBYTE" + 16 gerechnet. Falls der Sektor voll ist, wird der nächste Sektor angewählt und „DIRBYTE" wieder auf 0 gesetzt. Man kann bis zu 256 Einträge durchführen, dann erscheint die Fehlermeldung „Dir voll". Wird die maximale Datenmenge überschritten, so wird uns dies durch die Fehlermeldung „Disk Fehler" angezeigt.

Am Schluß wird, und das ist der Clou, der Bereich $9800 bis $9BFF auf den Sektor 1, Spur 0 geschrieben. Damit sind die Werte aller Variablen und darüber hinaus das Programm selbst automatisch gerettet. Das war auch der Grund, warum bei der Inbetriebnahme zunächst die Funktion „Speichern" aufgerufen werden mußte.

Wenn man weitere Disketten vorbereiten will, so kann man nun das Mikrodos mit „laden" in den Speicher holen. Um alle Variablen zu initialisieren wird es neu übersetzt, und dann kann man es auf die bekannte Weise auf beliebig viele, neu formatierte Disketten bringen. Das Mikrodos steht dann für den Aufruf per „Floppy-Start" bereit.

Das Programm stellt nur eine sehr einfache Möglichkeit für das Arbeiten mit der Floppy dar. Es bietet noch keinen Komfort. Aber es versetzt uns in die Lage, Programme speichern und laden zu können. Man könnte sich nun Hilfsprogramme auf Diskette ablegen, die z. B. einen Eintrag löschen oder eine Diskette kopieren.

```
dos4
fun.gos
modem
fisch 5

Dir=1 Laden=2 Speich.=3
```

Abb. 6.3.4
So sieht die Meldung aus, wenn eine Datei nicht da ist

```
Name:  hallo


nicht da
Frei: 757  K
```

Abb. 6.3.5 Aufbau eines Directory-Blocks

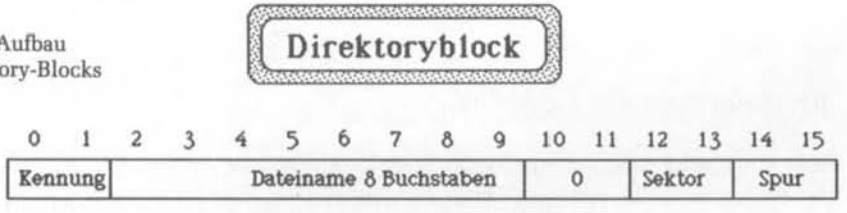
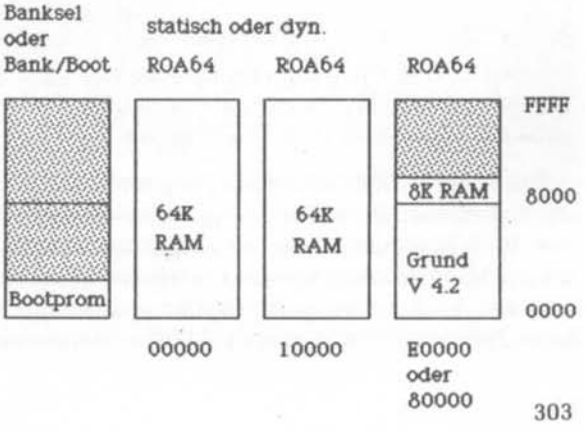


Abb. 6.4.1
Speicheraufteilung bei CP/M-68k



6.4 Das Betriebssystem CP/M-68k

Für Systeme mit den Prozessoren 68000 und 68008 gibt es auch ein komfortables Disketten-Betriebssystem: das CP/M-68k. CP/M ist die Abkürzung für „control program for microcomputer". Das CP/M wird bereits seit Jahren beim Z80 und 8080/8085 als Disketten-Betriebssystem verwendet. Es wird von zahlreichen Benutzern in aller Welt verwendet. CP/M wurde von der Firma Digital Research entwickelt und ist eines der wenigen Systeme, das anpaßbar ist. Das heißt, es ist möglich, CP/M und auch CP/M-68k auf verschiedene Hardware-Konfigurationen einzustellen. Alle hardwareabhängigen Programmteile, wie die Routinen für lesen und schreiben auf der Diskette oder die Ein- und Ausgabe von Zeichen, sind in einem speziellen Programmteil, BIOS genannt, zusammengefaßt. Dieses BIOS kann man selbst erstellen, man benötigt dazu allerdings einen CP/M-Computer. Für den NDR-KLEIN-Computer wurde diese Anpassung durchgeführt, das Ergebnis finden Sie auf den folgenden Seiten. Dabei kann die Beschreibung des CP/M-Betriebssystems nur knapp ausfallen, doch neben dem CP/M-68k-Betriebssystem ist auch ein umfangreiches Handbuch (s. Literaturverzeichnis) erhältlich. Für das angepaßte Betriebssystem sind ebenfalls Lieferadressen im Anhang genannt. CP/M-68k ist für zwei Diskettenformate lieferbar, einmal auf 8"-Disketten im Standard-CP/M-Format und dann auf 5¼"-Disketten im NDR80-Format. Das System wird auf etwa neun Disketten zu je ca. 250 KByte geliefert. Alles in allem fast 2 MByte Software. Neben dem Betriebssystem befindet sich ein C-Compiler auf den Disketten. C ist eine Programmiersprache, die Gleitkomma-Befehle enthält und sehr schnell ist. Ferner befinden sich ein Assembler, Bibliotheken und ein Editor, sowie viele weitere Dinge auf den Disketten.

Inbetriebnahme von CP/M-68k.

Abb. 6.4.1 zeigt eine schematische Speicheraufteilung für den Betrieb von CP/M-68k. Das Betriebssystem benötigt einen durchgehenden RAM-Speicher von Adresse 0 bis mindestens $1FFFF. Daher muß etwas an der Konfiguration geändert werden, denn unser Grundprogramm lag bisher ab Adresse 0 bis $7FFF. Nach dem Einschalten beginnt der Prozessor bei Adresse 0, es muß dort also zumindest nach dem Einschalten ein EPROM liegen, später wird dann in diesem Adreßbereich ein RAM angesprochen. Diesen Umschaltvorgang nennt man Banking. Die Bank/Boot-Baugruppe ist dafür erforderlich, auf der dafür eine spezielle Logik untergebracht ist. Das Grundprogramm wird auf eine ROA64-Baugruppe gesetzt und künftig in einem hohen Adreßbereich (z.B. ab $E0000) beheimatet sein. Als Arbeitsspeicher für das Grundprogramm muß man mindestens 8K, besser aber 16 KByte einsetzen.

Das Boot-EPROM enthält ein Programm, das automatisch die Bank/Boot-Baugruppe abschaltet, nach dem Grundprogramm sucht und dieses startet. Durch das Abschalten der Bank/Boot-Baugruppe wird der in diesem Bereich liegende RAM-Speicher ansprechbar. Das Grundprogramm legt dorthin automatisch seine TRAP- und Interruptvektoren, so daß man nun zunächst ganz normal mit dem Grundprogramm arbeiten kann. Der Adreßbereich 0 bis $1FFFF muß mit einem RAM versehen sein. Es kann in

ROA64-Baugruppen mit statischen Speichern ebenso wie mit der DYN64/256-Baugruppe und dynamischen RAM-Bausteinen realisiert sein. Wenn man später Programmiersprachen wie Fortran77, CBasic oder Pascal/MT+68k verwenden will, sollte man gleich 256 KByte Speicher vorsehen. Für das CP/M-68k und zum Betrieb des mitgelieferten C-Compilers genügen jedoch 128KByte.

Abb. 6.4.2 zeigt die Anordnung auf der Bank/Boot-Baugruppe. Das „BOOTR"-EPROM enthält nur wenige Bytes, weshalb es auch manchmal in einem 2716-EPROM geliefert wird. Die Brücken rechts unten muß man auftrennen, sie sind für den Z80-Betrieb vorgesehen. Darunter steht auf der Leiterplatte auch „Z80". Die anderen Brücken bleiben wie voreingestellt (siehe Kapitel 7).

Auf der FLO2-Baugruppe muß man auch eine Veränderung vornehmen. *Abb. 6.4.3* zeigt ein Schema. Die Disketten-Software im Grundprogramm arbeitet ohne Interrupts, daher sollte man die Interrupt-Leitung, die von der FLO2-Baugruppe zum Bus führt, trennen. Das CP/M-68k-Betriebssystem gibt alle Interrupts frei, es könnte somit zu Störungen (Meldung „INTERRUPT") kommen. Die Leitung läßt sich auf der Lötseite der FLO2-Baugruppe gut erreichen und leicht trennen, sie ist dort mit „INT" beschriftet.

Dann kann es losgehen. Nach dem Einlegen der System-Diskette, die auch das Start-Programm auf den System-Spuren enthält, wird „Floppy-Start" im Grundprogramm-Menue aufgerufen. Die *Abb. 6.4.4* soll nach einer Weile und einigen Disketten-Zugriffen auf dem Bildschirm erscheinen. CP/M-68k ist dann betriebsbereit.

Wie arbeitet CP/M?

Zunächst arbeitet CP/M wesentlich komfortabler als unser bisheriges Mikrodos. Dort wurde nur der Startsektor einer Datei angeben. Möchte man eine Datei löschen, um den Diskettenplatz wieder zu gewinnen, müßte man alle nachfolgenden Dateien verschieben, das ist sehr aufwenig. Im CP/M hat man sich dazu etwas trickreiches ausgedacht. Man schreibt in den Directory-Block Blocknummern; jeder Block entspricht einem Sektor, 16 Blöcke bringt man in einem Directory-Block unter (*Abb. 6.4.5*). Wenn 16 K belegt sind, wird eine Datei mit dem gleichen Namen, aber einer neuen sogenannten Extensionnummer angelegt. So weiß man später, welche Blöcke zu einer Datei gehören. Eine Recordzahl gibt noch in 128-Byte-Schritten an, wieviel Platz belegt ist. Am Anfang der Eintragung gibt außerdem eine Kennung darüber Auskunft, ob der Eintrag gültig ist. Ein solcher Block belegt 32 Byte, so daß man für 64 Einträge 2048 Byte benötigt. Wenn man eine Datei löscht, so werden alle belegten Blöcke wieder verfügbar. Dazu wird eine sogenannte Allocate-Map geführt, eine Bit-Tabelle, in der ein gesetztes Bit einem belegten Block entspricht. Dadurch, daß die Blöcke möglichst groß gewählt werden, minimal 1 K, meist 2 K oder 4 K, ist die Anzahl der Bits in dieser Tabelle nicht so sehr groß. Sie hängt natürlich auch von der Diskettenkapazität ab. Beim 8"-Format verwenden wir 1-K-Blöcke und 64 Directory-Einträge, beim NDR80-Format stehen 2 K pro Block und 256 Directory-Einträge zur Verfügung.

Abb. 6.4.6 zeigt die Disketten-Information, wie man sie bei Benutzung des STAT-Programms unter CP/M-68k erhält. Hier für das NDR80-Format und die Ramfloppy. Eine Ramfloppy verhält sich, aus der Sicht des Benutzers betrachtet, wie ein Disketten-

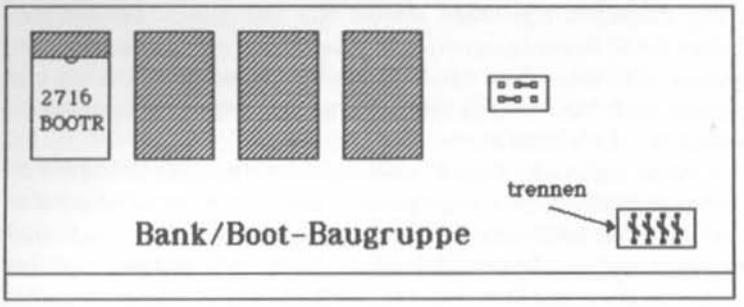


Abb. 6.4.2
Die Belegung der BANK/BOOT-Baugruppe

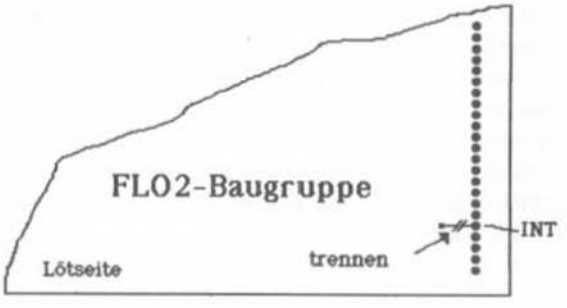


Abb. 6.4.3
Die FLO2 wird umgerüstet

```
CP/M-68K(tm) Version 1.2  03/20/84
Copyright (c) 1984 Digital Research, Inc.

A>
```

Abb. 6.4.4
So meldet sich CP/M-68k

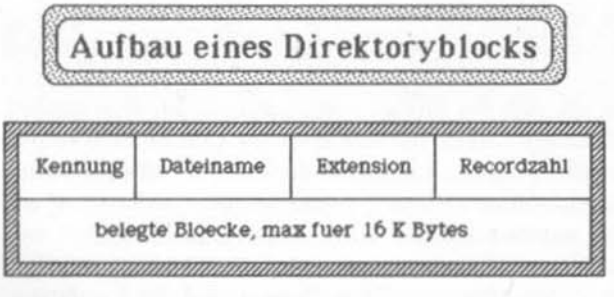


Abb. 6.4.5 Aufbau eines CP/M-68k-Direktory-Blocks

```
A>
A>stat dsk:

       A: DRIVE CHARACTERISTICS
   6,224: 128 BYTE RECORD CAPACITY
     778: KILOBYTE DRIVE  CAPACITY
     256: 32 BYTE  DIRECTORY ENTRIES
     256: CHECKED  DIRECTORY ENTRIES
     128: 128 BYTE RECORDS / DIRECTORY ENTRY
      16: 128 BYTE RECORDS / BLOCK
      40: 128 BYTE RECORDS / TRACK
       4: RESERVED  TRACKS

       G: DRIVE CHARACTERISTICS
   2,048: 128 BYTE RECORD CAPACITY
     256: KILOBYTE DRIVE  CAPACITY
      64: 32 BYTE  DIRECTORY ENTRIES
       0: CHECKED  DIRECTORY ENTRIES
     128: 128 BYTE RECORDS / DIRECTORY ENTRY
       8: 128 BYTE RECORDS / BLOCK
     256: 128 BYTE RECORDS / TRACK
       0: RESERVED  TRACKS

A>
```

Abb. 6.4.6 Ausgabe nach dem STAT-Befehl

```
...A68K 4.0a 06/13/83   ...Run on 04/19/85
                          1. ************************************
                          2. * Boot fuer Bankboot/ Boot-Karten  *
                          3. * (C) 1984 Rolf-Dieter Klein       *
                          4. * V1.0 841016                     *
                          5. ************************************
                          6. * 68008 Boot
                          7.
)00000                    8.          section 0
                          9.
                         10.
)00000  00008FFE         11.          dc.l $8ffe       * dummy Stackpointer
)00004  00000008         12.          dc.l start       * Start 1. Stufe
                         13.
                         14.
)00008  363C 0031        15. start:   move #ende-anf-1,d3
)0000C  41F9 00008000    16.          lea $8000,a0     * Ziel-Gebiet fuer Kalt-Start
)00012  43FA 000E        17.          lea anf(pc),a1   * Quelle bei Start
)00016                   18. transport:
)00016  10D9             19.          move.b (a1)+,(a0)+
)00018  51CB FFFC        20.          dbra d3,transport
)0001C  4EF9 00008000    21.          jmp $8000        * Und ins Zielspringen
                         22.
                         23.
)00022                   24. anf:
                         25.                           * Suchen nach ROM-Gebiet
                         26.                           * auf Adresse 0
)00022  1038 7000        27. boot:    move.b $7000,d0 * Loeschen Boot
)00026  11FC 0080        28.          move.b #$80,$fffffc8  * loeschen Bankboot
        FFC8
)0002C  41F8 0400        29.          lea $400,a0      * Start der Suche
)00030  0C90 5AA58001    30. loop:    cmp.l #$5aa58001,(a0)
)00036  6708             31.          beq.s gefunden
)00038  D1FC 00001000    32.          adda.l #$1000,a0 * Alle 4K Suchen nach dem Anfang
)0003E  60F0             33.          bra.s loop
)00040                   34. gefunden:
)00040  0C68 6000        35.          cmp #$6000,$20(a0)
        0020
)00046  66E8             36.          bne.s loop              * dort muss Sprung stehen
)00048  0C68 6000        37.          cmp #$6000,$24(a0)      * dort muss Sprung stehen
        0024
)0004E  66E0             38.          bne.s loop              * ok Test scheint ok
)00050  4EE8 0024        39.          jmp $24(a0)             * also Cold-Start ausfueh
)00054                   40. ende:
                         41.
)00054                   42.          end
    0 Errors detected
```

Abb. 6.4.7
Der Inhalt des BOOT-EPROMs

Abb. 6.4.8 So kann man das Grundprogramm verschieben

```
start:                  * Grundprog. auf 0
 lea $e0000,a0          * Quelle
 lea 0,a1               * Ziel
 move #$2000-1,d3       * Anzahl Langworte
schleife:               * Wiederhole
 move.l (a0)+,(a1)+     * Grundprog. Transp.
 dbra d3,schleife       * und transportieren
 jmp $424               * Kalt-Start Grundp.
```

Laufwerk, in Wirklichkeit liegen die Daten aber in einem RAM. Bei unserem CP/M-68k kann man ein RAM mit 256 KByte als Laufwerk verwenden. Der Vorteil besteht in der wesentlich höheren Arbeitsgeschwindigkeit gegenüber einem Laufwerk. Für die Unterbringung dieses Speichers wird eine zusätzliche RAM-Baugruppe (1x DYN64/256 mit 256 KByte oder 4x ROA64 mit insgesamt 256 KByte) ab Adresse $80000 bis $BFFFF benötigt. Das Grundprogramm muß dann natürlich dahinter liegen.

CP/M arbeitet mit Sektoren, die 128 Byte lang sind. Wenn man Disketten mit einem anderen Format verwendet, so muß man die Sektoren entsprechend aufteilen. Das geschieht im BIOS automatisch, z.B. wenn das NDR80-Format mit 1024 Byte pro Sektor verwendet wird.

Nun noch ein paar praktische Hinweise und Informationen zum Betrieb. *Abb. 6.4.7* zeigt den Inhalt des „BOOTR"-EPROMs. Das Programm transportiert zunächst den zwischen den Marken „boot" und „ende" liegenden Maschinencode in den RAM-Speicher ab Adresse $8000. Anschließend wird dieses Programm durch einen Sprung zu dessen Anfang gestartet. Zunächst schaltet die Routine die Bank/Boot-Baugruppe ab. Die Eingabe von Adresse $7000 dient zum Löschen einer älteren Baugruppe, der Banksel-Baugruppe, die auch verwendet werden kann. Der Port $FFFFFFC8 dient zum Umschalten der Bank/Boot-Baugruppe und wird mit $80 belegt, die Karte wird damit inaktiv.
Dann wird nach dem Datenmuster $5aa58001 gesucht. Dies ist eine Kennung im Grundprogramm-EPROM. Daraus läßt sich die Startposition berechnen.

Wenn Sie das Grundprogramm einmal im Bereich ab der Adresse 0 benötigen, z.B. weil bestimmte Programme nur dort laufen können, so ist das kein Problem. Das Programm der *Abb. 6.4.8* transferiert das Grundprogramm in diesen Bereich. Wenn Sie die EPROMs nicht ab Adresse $E0000 stehen haben, so wie es das Verschiebeprogramm voraussetzt, müssen Sie als „Quelle" Ihre Adresse einsetzen. Das Programm startet das verschobene Grundprogramm dann automatisch.

Wenn Sie anschließend den RESET-Taster drücken, so meldet sich immer das Grundprogramm, das auf der niedrigsten Adresse sitzt, da das „BOOTR"-EPROM immer von unten anfängt, das Grundprogramm zu suchen. Um das Grundprogramm im EPROM wieder zu starten, geht man in das Menue „Starten" und gibt die Adresse $E0424 an.

Beim Booten des CP/Ms geschieht folgendes: Zunächst wird der erste Sektor auf der Diskette geladen, also Spur 0, Sektor 1. Er beinhaltet ein weiteres Ladeprogramm, „ndrboot" genannt. *Abb. 6.4.9* zeigt das Listing dazu.

Dieses Programm besteht eigentlich aus zwei Teilen, einem Boot für das 8"-Format (Marke „maxiboot") und einem für das 5¼"-Format (Marke „miniboot"). Welches Format vorliegt, wird durch das Programm automatisch festgestellt.

Es werden dann die restlichen Sektoren mit dem eigentlichen CP/M-68k geladen, dabei handelt es sich aber zunächst um eine verkürzte Form des Betriebssystems; die endgültige Variante wird erst durch einen weiteren Ladevorgang eingelesen.

Das CP/M-68k findet sich samt BIOS in der Datei „CPM.SYS". *Abb. 6.4.10* zeigt das vollständige BIOS, in dem nun alle Fähigkeiten vorhanden sind. Wie man dieses BIOS ändert und in das CPM.SYS einbaut, ist ausführlich im CP/M-68k-Handbuch beschrieben und würde hier zu weit führen. Sie finden auf unserer Diskette auch noch ein

```
C P / M   6 8 0 0 0   A s s e m b l e r                Revision 04.03                Page
Source File: ndrboot.s

   1                                ********************************************
   2                                * Boot Load- Routine fuer Autostart         *
   3                                * durch Grundprogramm                       *
   4                                * dahinter befindet sich direkt das CP/M    *
   5                                * Betriebsystem                             *
   6                                * V 1.0    Rolf-Dieter Klein                *
   7                                * Boot 8" // 5 1/4" mit autom. Erkennung    *
   8                                ********************************************
   9
  10                                tpa equ $400   * Zieladresse, auch bei LOAD angeben.
  11
  12                                start:
  13 00000000 4E71                   nop      * Erkennung fuer Grundprogramm, das gueltiger Bo
da
  14 00000002 7E14                   move #20,d7 * CLRSCREEN
  15 00000004 4E41                   trap #1  * Bildschirm loeschen
  16 00000006 7011                   move #$11,d0
  17 00000008 7E19                   move #25,d7 * SIZE
  18 0000000A 4E41                   trap #1  * Schriftgroesse auf 80 Zeichen setzen
  19 0000000C 7801                   move #$1,d4
  20 0000000E 7E4C                   move #76,d7
  21 00000010 4E41                   trap #1  * Floppy Erkennung, d4=Laufwerkscode
  22                                *
  23 00000012 B83C0021               cmp.b #$21,d4
  24 00000016 6748                   beq miniboot
  25 0000001B B83C0011               cmp.b #$11,d4 * maxi-code
  26 0000001C 6702                   beq maxiboot
  27 0000001E 4E75                   rts      * System Fehler
  28
  29
  30                                maxiboot: * 26 Sektoren 2 Spuren
  31 00000020 363C0083               move #$83,d3 * Steprate slow, SSD
  32 00000024 4241                   clr d1       * setzen
  33 00000026 7E4B                   move #75,d7
  34 00000028 4E41                   trap #1
  35 0000002A 41F900000400           lea tpa,a0 * Startadresse
  36 00000030 7600                   move #0,d3 * Spur 0
  37                                ladem0:
  38 00000032 7401                   move #1,d2 * Sektor 1
  39                                ladem1:
  40 00000034 7201                   move #1,d1 * LESEN
  41 00000036 7811                   move #$11,d4 * Laufwerk A SD, 8"
  42 00000038 48E77880               movem.l d1-d4/a0,-(a7)
  43 0000003C 7E4B                   move #75,d7
  44 0000003E 4E41                   trap #1
  45 00000040 4CDF011E               movem.l (a7)+,d1-d4/a0
  46 00000044 D1FC00000080           adda.l #128,a0 * 128 Bytes pro Sektor
  47 0000004A 5242                   add #1,d2
  48 0000004C B47C001A               cmp #26,d2
  49 00000050 6FE2                   ble ladem1  * 1,2,...26 erlabt
  50 00000052 5243                   add #1,d3   * Spuren 0,1
  51 00000054 B67C0001               cmp #1,d3
  52 00000058 6FD8                   ble ladem0  * dann fertig
  53 0000005A 4EF9000000AE           jmp cpm     * muss absolute sein, da boot nicht
  54                                *              auf entgueltiger Adresse steht
  55
  56                                miniboot:     * 5 Sektoren, zweiseitig, 2 Spuren laden
  57 00000060 363C0083               move #$83,d3 * Steprate slow, SSD
  58 00000064 4241                   clr d1       * setzen
  59 00000066 7E4B                   move #75,d7
  60 00000068 4E41                   trap #1
```

zu Abb. 6.4.9

```
C P / M   6 8 0 0 0   A s s e m b l e r                Revision 04.03      Page   2
Source File: ndrboot.s

  61 0000006A 41F900000400          lea tpa,a0 * Startadresse
  62 00000070 7600                  move #0,d3 * Spur 0
  63                               ladex0:
  64 00000072 7401                  move #1,d2 * Sektor 1
  65                               ladex1:
  66 00000074 7201                  move #1,d1 * LESEN
  67 00000076 7821                  move #$21,d4 * Laufwerk A DD  5 1/4"
  68 00000078 48E77880              movem.l d1-d4/a0,-(a7)
  69 0000007C B47C0006              cmp #6,d2
  70 00000080 6D06                  blt skipx
  71 00000082 5B42                  sub #5,d2  * 1..5
  72 00000084 887C0080              or #$80,d4 * Rueckseite
  73                               skipx:
  74 00000088 7E4B                  move #75,d7
  75 0000008A 4E41                  trap #1
  76 0000008C 4CDF011E              movem.l (a7)+,d1-d4/a0
  77 00000090 D1FC00000400          adda.l #1024,a0 * 1024 Bytes pro Sektor
  78 00000096 5242                  add #1,d2
  79 00000098 B47C000A              cmp #10,d2
  80 0000009C 6FD6                  ble ladex1  * 1,2,...10 erlabt (6..10 = Rueckseite)
  81 0000009E 5243                  add #1,d3   * Spuren 0,1
  82 000000A0 B67C0001              cmp #1,d3
  83 000000A4 6FCC                  ble ladex0  * dann fertig
  84 000000A6 4EF9000000AE          jmp cpm
  85 000000AC 4E71                  nop
  86
  87
  88                               cpm:      * Start des CP/Ms hier.
  89 000000AE 4E71                  nop
  90
  91 000000B0                       end
```

Abb. 6.4.9 Der Boot-Lader des CP/M-68k

```
C P / M   6 8 0 0 0   A s s e m b l e r                Revision 04.03      Page   1
Source File: ndrbios.s

   1                    ********************************************
   2                    * BIOS 68k 1.2 fuer  NDR-Klein-Computer     *
   3                    * Version 2.0  // ROM //   R-D.Klein 850321 *
   4                    * // 8 " Laufwerke // 5 1/4 80 Spur //      *
   5                    * RAM-Floppy                                *
   6                    * Version fuer AS68      850323    1.0      *
   7                    * CPM.SYS  erzeugen                         *
   8                    * dazu  AS68 ndrbios.s                      *
   9                    * LO68 -r -ucpm cpm.rel cpmlib ndrbios.o    *
  10                    * RELOC -B19000 cpm.rel cpm.sys             *
  11                    * dann auf Systemdiskette copieren.         *
  12                    * bei AS68 und LO68 ggf. Laufwerke angeben  *
  13                    * ebenfalls mit -s x: die Init-Datei, siehe *
  14                    * CP/M Manual                                *
  15                    ********************************************
  16
  17                     .globl _init                          zu Abb. 6.4.10
  18                     .globl _ccp
  19                     .globl cpm     * Startadresse CP/M (erste Adresse)
```

```
20                              *               Damit ist das Bios,
21                              *               und damit CPM.REL verschiebbar.
22                              * man mu~ nur bei RELOC eine neue Adresse angeben
23                              * bei 128K  -B19000  , Kontrolle durch SIZE68 moeglich.
24                              * bei 256K  -B39000
25                              *
26                              * CP/M Start, bei reloc
27                              * angeben, und auch hier ggf.
28                              * aendern, sonst reagiert das cp/m nicht
29                              * denn es holt sich die Information aus
30                              * der Memory region table
31                              * kontrolle des wertes durch size68 durchfuehren
32                              * wenn bios geaendert wird.
33                              *
34                              * 8 Zoll Version  // wird automatisch eingestellt
35                              *               // abhaengig von der Diskette
36                              * Laufwerk   A,B,C,D       8 Zoll je 243K A,C = Vorderseite
37                              *            E,F           5 1/4    je 800K
38                              *            G             Ramfloppy
39                              *
40                              * 5 1/4 Zoll Version // wird automatisch eingestellt
41                              *                    // abhaengig von der Diskette
42                              * Laufwerk   A,B           5 1/4 " je 800K
43                              *            C,D,E,F       8 " je 243K C,E=Vorderseite
44                              *            G             Ramfloppy
45                              *
46                              * 8 "    koennen zweiseitig sein,
47                              *        aber einfache Dichte, 128 Bytes / Sektor
48                              *        mit 26 Sektoren pro Spur und 77 Spuren
49                              *
50                              * 5 1/4" (auch 3 1/2" oder 3")
51                              *        muessen zweiseitig sein, 80 Spuren haben
52                              *        und 1024 Bytes pro Sektor mit 5 Sektoren pro Spur
53                              *        ( doppelte Dichte)
54                              *
55                              *
```

```
C P / M   6 8 0 0 0   A s s e m b l e r                Revision 04.03              Page   2
Source File: ndrbios.s

56                              * die Ramfloppy ist fuer 256 K Byte ausgelegt
57                              * sie beginnt bei der Adresse RAMFLO und
58                              * mu~ vor Benutzung mit E5 vorgeloescht werden
59                              * ein ERA *.* genuegt nicht.
60                              * Ein BAD SEKTOR tritt auf, wenn ein Fehler
61                              * auf der RAM-Karte festgestellt wurde. Dabei
62                              * wird aber nur grob geprueft.
63                              *
64                              ramflo equ $80000 * Ramfloppy Startadr.
65                              *
66                              *
67                              * I/O Byte wird ausgewertet
68                              * 7     6     5     4     3     2     1     0
69                              * -list-      -auxo-      -auxi-      -cons--
70                              *
71                              *
72                              * cons:  *00=key/gdp 01=si/so 10=(auxi,list) 11= key/gdp
73                              * auxi:   00=key     *01=si     10=ri          11= key
74                              * auxo:   00=gdp     *01=so     10=po          11= gdp
75                              * list:   00=gdp      01=so    *10=lo          11= gdp
76                              *    * = Default nach Start
77                              * andere Anordnungen durch STAT programmieren
78                              * siehe CP/M68 Manual.
```

zu Abb. 6.4.10

```
 79
 80
 81                                  ****************************************************
 82
 83                                  _init:
 84                                  *                   zuerst Grundprogramm suchen
 85                                  *                   aber erst hinter der TPA
 86 00000000 41F900020400             lea $20400,a0 * Start der Suche 128K RAM davor annehmen
 87                                  loop:
 88 00000006 0C905AA58001             cmp.l #$5aa58001,(a0)  * Kennung im Grundprogramm
 89 0000000C 6708                     beq.s gefunden
 90 0000000E D1FC00001000             adda.l #$1000,a0       * Alle 4K Suchen nach dem Anfang
 91 00000014 60F0                     bra.s loop
 92                                  gefunden:
 93 00000016 0C6860000020             cmp #$6000,$20(a0)
 94 0000001C 66E8                     bne.s loop             * dort muss Sprung stehen
 95 0000001E 0C6860000024             cmp #$6000,$24(a0)     * dort muss Sprung stehen
 96 00000024 66E0                     bne.s loop             * ok Test scheint ok
 97                                  *                        a0 -> Grundprog + $400
 98 00000026 91FC00000400             suba.l #$400,a0        * BASIS-Adresse
 99 0000002C 23C800000016             move.l a0,gru68k       * dort merken
100 00000032 D1FC00000420             adda.l #$420,a0        * Adresse fuer Trap-Sprung
101 00000038 23C80000001A             move.l a0,gruj68k      * auch merken
102                                  * Bildschirm und Size sind im Boot vorbereitet
103                                  *                        Laufwerks-Dichte bestimmen
104 0000003E 7801                     move #1,d4             * Laufwerk A
105 00000040 7E4C                     move #76,d7            * GETFLOP
106 00000042 610006A8                 bsr trapexe
107 00000046 13C400000004             move.b d4,drvcode      * $11=Mini DD, $21=Maxi SD
108                                  *
109 0000004C 423900000014             clr.b mwrtflg          * alles voreinstellen
110 00000052 13FC00FF0000000F         move.b #$ff,lastsso    * damit wird Steprate zuerst eingest.


C P / M   6 8 0 0 0   A s s e m b l e r                Revision 04.03              Page  3
Source File: ndrbios.s

111 0000005A 423900000010             clr.b mdrvakt          * 0=ungueltiges Laufwerk in Buffer
112 00000060 427900000006             clr cursor             * Cursor ist jetzt aus
113 00000066 13FC000000000000         move.b #0,stepmaxi     * PATCH PATCH PATCH maxi-Steprate
114 0000006E 13FC000000000001         move.b #0,stepmini     * PATCH PATCH PATCH mini-Steprate
115                                  *                        Rest vorbereiten
116 00000076 103C001E                 move.b #$1e,d0         * 9600 Baud, 8 Bit, 1 Stopbit
117 0000007A 123C000B                 move.b #$0b,d1         * keine Paritaet, RTS high, T,R Freig
abe.
118 0000007E 7E6C                     move #108,d7           * SIINIT
119 00000080 6100066A                 bsr trapexe            * SER-Baugruppe fuer AUX (RDR,PUN).
120                                  *
121 00000084 33FC009400000002         move.w #%10010100,iobyte * LO, SI/SO, KEY/GDP einstellen.
122 0000008C 23FC0000009A0000008C     move.l #traphndl,$8c   * trap #3 handler
123 00000096 4280                     clr.l d0               * Laufwerk A, User 0 ist Start
124 00000098 4E75                     rts
125
126                                  traphndl:
127 0000009A 0C400018                 cmpi #nfuncs,d0
128 0000009E 6410                     bcc trapng
129 000000A0 E548                     lsl #2,d0
130 000000A2 48C0                     ext.l d0
131 000000A4 D0BC000000B2             add.l #biosbase,d0
132 000000AA 2040                     movea.l d0,a0
133 000000AC 2050                     movea.l (a0),a0
134 000000AE 4E90                     jsr (a0)
135                                  trapng:
```

zu Abb. 6.4.10

```
136 000000B0 4E73                rte
137
138                              biosbase:
139 000000B2 00000000            dc.l _init     * 0
140 000000B6 0000011A            dc.l wboot     * 1
141 000000BA 0000012A            dc.l constat   * 2
142 000000BE 00000154            dc.l conin     * 3
143 000000C2 00000174            dc.l conout    * 4
144 000000C6 000001D8            dc.l lstout    * 5
145 000000CA 000001B6            dc.l pun       * 6
146 000000CE 00000194            dc.l rdr       * 7
147 000000D2 000002D8            dc.l home      * 8
148 000000D6 000002E0            dc.l seldsk    * 9
149 000000DA 000002FE            dc.l settrk    * 10
150 000000DE 00000306            dc.l setsec    * 11
151 000000E2 00000324            dc.l setdma    * 12
152 000000E6 0000035C            dc.l read      * 13
153 000000EA 00000396            dc.l write     * 14
154 000000EE 000001F8            dc.l listst    * 15
155 000000F2 0000030E            dc.l sectran   * 16
156 000000F6 00000324            dc.l setdma    * 17
157 000000FA 00000688            dc.l getseg    * 18
158 000000FE 00000690            dc.l getiob    * 19
159 00000102 00000698            dc.l setiob    * 20
160 00000106 00000672            dc.l flush     * 21
161 0000010A 000006A0            dc.l setexc    * 22
162 0000010E 00000112            dc.l getbasis  * 23   neuer Bios-Eingang. a4 belegen mit BA!
IS
163
164                              nfuncs equ (*-biosbase)/4
165


C P / M   6 8 0 0 0   A s s e m b l e r                Revision 04.03              Page   4
Source File: ndrbios.s

166                              getbasis:               * BIOS-Eingang, nur fuer Grundprg. O|
erationen
167 00000112 287900000016        move.l gru68k,a4        * aus Hilfszelle holen und in A4 als
Ergebnis.
168 00000118 4E75                rts
169
170                              wboot:
171 0000011A 61000492            bsr puttrk              * falls noch aussteht
172 0000011E 423900000010        clr.b mdrvakt           * und Laufwerk ungueltig danach
173 00000124 4EF900000000        jmp _ccp
174
175                              constat:
176 0000012A 303900000002        move iobyte,d0
177 00000130 C07C0003            and #3,d0
178 00000134 670000E4            beq csts          * 00 = key
179 00000138 B07C0001            cmp #1,d0
180 0000013C 6700017C            beq sists         * 01 = si
181 00000140 B07C0002            cmp #2,d0
182 00000144 6704                beq allfalse      * 10 = batch, auxi, kein Status
183 00000146 600000D2            bra csts          *
184
185                              allfalse:         * kein Status bei BATCH
186 0000014A 4240                clr d0            * nicht abbrechbar
187 0000014C 4E75                rts
188
189
190
```

zu Abb. 6.4.10

```
 191                                alltrue:
 192 0000014E 303C00FF               move #$00ff,d0
 193 00000152 4E75                   rts
 194
 195                                conin:
 196 00000154 303900000002           move iobyte,d0
 197 0000015A C07C0003               and #3,d0
 198 0000015E 670000E6               beq ci              * 00 = key
 199 00000162 B07C0001               cmp #1,d0
 200 00000166 6700014A               beq si              * 01 = si
 201 0000016A B07C0002               cmp #2,d0
 202 0000016E 6724                   beq rdr             * 10 = batch, auxi
 203 00000170 600000D4               bra ci              *
 204
 205
 206                                conout:
 207 00000174 303900000002           move iobyte,d0
 208 0000017A C07C0003               and #3,d0
 209 0000017E 670000E8               beq co              * 00 = gdp
 210 00000182 B07C0001               cmp #1,d0
 211 00000186 67000114               beq so              * 01 = so
 212 0000018A B07C0002               cmp #2,d0
 213 0000018E 6748                   beq lstout          * 10 = batch, auxi
 214 00000190 600000D6               bra co              *
 215
 216
 217                                rdr:
 218 00000194 303900000002           move iobyte,d0
 219 0000019A C07C000C               and #%00001100,d0
 220 0000019E 670000A6               beq ci              * 00 = key

C P / M   6 8 0 0 0   A s s e m b l e r          Revision 04.03          Page   5
Source File: ndrbios.s

 221 000001A2 B07C0004               cmp #%00000100,d0
 222 000001A6 6700010A               beq si              * 01 = si
 223 000001AA B07C0008               cmp #%00001000,d0
 224 000001AE 67000116               beq ri              * 10 = ri
 225 000001B2 60000092               bra ci              *
 226
 227
 228                                pun:            * Ausgabe nur dann, wenn DSR auf
 229 000001B6 303900000002           move iobyte,d0
 230 000001BC C07C0030               and #%00110000,d0
 231 000001C0 670000A6               beq co              * 00 = gdp
 232 000001C4 B07C0010               cmp #%00010000,d0
 233 000001C8 670000D2               beq so              * 01 = so
 234 000001CC B07C0020               cmp #%00100000,d0
 235 000001D0 670000FC               beq po              * 10 = po
 236 000001D4 60000092               bra co              *
 237
 238
 239                                lstout:
 240 000001D8 303900000002           move iobyte,d0
 241 000001DE C07C00C0               and #%11000000,d0
 242 000001E2 67000084               beq co              * 00 = co
 243 000001E6 B07C0040               cmp #%01000000,d0
 244 000001EA 670000B0               beq so              * 01 = so
 245 000001EE B07C0080               cmp #%10000000,d0
 246 000001F2 67000092               beq lo              * 10 = lo
 247 000001F6 6070                   bra co
 248
 249
```

zu Abb. 6.4.10

```
250                                 liststst:
251 000001F8 303900000002           move iobyte,d0
252 000001FE C07C00C0               and #%11000000,d0
253 00000202 6700FF4A               beq alltrue       * 00 = co, immer bereit
254 00000206 B07C0040               cmp #%01000000,d0
255 0000020A 6700009A               beq ssts          * 01 = so
256 0000020E B07C0080               cmp #%10000000,d0
257 00000212 6700007C               beq lssts         * 10 = lo
258 00000216 6000FF36               bra alltrue       * Ausgabe immer bereit
259
260
261
262                                 csts:
263 0000021A 4A7900000006           tst cursor
264 00000220 660E                   bne conistat          * Cursor ein, dann skip
265 00000222 33FC000100000006       move #1,cursor
266 0000022A 7E3D                   move #61,d7
267 0000022C 610004BE               bsr trapexe     * CURSEIN
268                                 conistat:
269 00000230 7E3C                   move #60,d7
270 00000232 610004B8               bsr trapexe     * Autoflip
271 00000236 7E0D                   move #13,d7
272 00000238 610004B2               bsr trapexe     * CSTS
273 0000023C 6704                   beq.s noton
274 0000023E 7001                   moveq.l #$1,d0
275 00000240 4E75                   rts
```



```
276                                 noton:
277 00000242 42B0                   clr.l d0
278 00000244 4E75                   rts
279
280
281                                 ci:
282 00000246 4A7900000006           tst cursor
283 0000024C 670C                   beq conin
284 0000024E 427900000006           clr cursor
285 00000254 7E3E                   move #62,d7
286 00000256 61000494               bsr trapexe     * CURSAUS
287                                 conin:
288 0000025A 7E0C                   move #12,d7
289 0000025C 6100048E               bsr trapexe
290 00000260 C0BC0000007F           and.l #$7f,d0   * Wort wird ausgewertet
291 00000266 4E75                   rts
292
293
294                                 co:
295 0000026B 4A7900000006           tst cursor
296 0000026E 670C                   beq conout
297 00000270 427900000006           clr cursor
298 00000276 7E3E                   move #62,d7
299 00000278 61000472               bsr trapexe     * CURSAUS
300                                 conout:
301 0000027C 3001                   move d1,d0
302 0000027E 7E21                   move #33,d7
303 00000280 6100046A               bsr trapexe
304 00000284 4E75                   rts
305
306
```

zu Abb. 6.4.10

```
307                                 lo:
308 00000286 3001                    move d1,d0
309 00000288 7E16                    move #22,d7     * LO
310 0000028A 61000460                bsr trapexe
311 0000028E 4E75                    rts
312
313                                 losts:
314 00000290 7E75                    move #117,d7    * LSTS
315 00000292 61000458                bsr trapexe     * 0=nicht ready, $00ff=ready
316 00000296 C07C00FF                and #$ff,d0     * savety
317 0000029A 4E75                    rts
318
319                                 so:
320 0000029C 3001                    move d1,d0      * HIGH liegt (12V).
321 0000029E 7E69                    move #105,d7    * SO, Ausgabe seriell
322 000002A0 6100044A                bsr trapexe     * d0.b bleibt Zeichen
323 000002A4 4E75                    rts
324
325                                 sosts:
326 000002A6 7E6B                    move #107,d7
327 000002A8 61000442                bsr trapexe     * FF = True
328 000002AC C07C00FF                and #$ff,d0
329 000002B0 4E75                    rts
330
```



```
331                                 si:
332 000002B2 7E6B                    move #104,d7    * SI, Eingabe seriell
333 000002B4 61000436                bsr trapexe     * d0.b = Zeichen
334 000002B8 4E75                    rts
335
336                                 sists:
337 000002BA 7E6A                    move #106,d7    * SISTS, FFFFFFFF=Zeichen da
338 000002BC 6100042E                bsr trapexe
339 000002C0 C07C00FF                and #$ff,d0     * nur FF wenn TRUE
340 000002C4 4E75                    rts
341
342                                 ri:
343 000002C6 7E0E                    move #14,d7     * RI
344 000002C8 61000422                bsr trapexe
345 000002CC 4E75                    rts
346
347                                 po:
348 000002CE 3001                    move d1,d0
349 000002D0 7E0F                    move #15,d7     * PO
350 000002D2 61000418                bsr trapexe
351 000002D6 4E75                    rts
352
353
354                                 home:
355 000002D8 42390000000C            clr.b track
356 000002DE 4E75                    rts
357
358                                 seldsk:         * je nach Boot-Quelle
359 000002E0 7000                    moveq #0,d0
360 000002E2 B23C0007                cmp.b #maxdsk,d1 * max disk
361 000002E6 6A14                    bpl selrtn
362 000002E8 13C10000000E            move.b d1,seldrv
363 000002EE 610003C0                bsr convdrv     * nach d0, Langwort 0,1,2,3,4,5,6
364                                 *                 unabh. von Boot 0..3 = maxi, 4..5=mini
```

zu Abb. 6.4.10

```
365 000002F2 C0FC001A              mulu #dphlen,d0 * v 2.0 alter Fehler behoben.
366 000002F6 D0BC00000028          add.l #dph0,d0
367                               selrtn:
368 000002FC 4E75                  rts
369
370                               settrk:
371 000002FE 13C10000000C          move.b d1,track
372 00000304 4E75                  rts
373
374                               setsec:
375 00000306 13C10000000D          move.b d1,sector  * 1..256 moegl. 256->0 abgebildet
376 0000030C 4E75                  rts
377
378                               sectran:
379 0000030E 4A82                  tst.l d2
380 00000310 670C                  beq nosect
381 00000312 2042                  movea.l d2,a0
382 00000314 48C1                  ext.l d1
383 00000316 10301000              move.b 0(a0,d1),d0
384 0000031A 48C0                  ext.l d0
385 0000031C 4E75                  rts
```

```
C P / M   6 8 0 0 0   A s s e m b l e r                Revision 04.03              Page   8
Source File: ndrbios.s

386                               nosect:
387 0000031E 3001                  move d1,d0
388 00000320 5240                  add #1,d0       * 1..256 / 1..40 / besser fuer allg.
389 00000322 4E75                  rts             * Routinen, wie PUTBOOT, da in Manual so.
390
391
392                               setdma:
393 00000324 23C100000008          move.l d1,dma
394 0000032A 4E75                  rts
395
396                               dcode:          * Codieren Drive + Dense
397 0000032C 61000382              bsr convdrv     * d0= 0,1,2,3  immer. Laufwerkscode
398 00000330 C07C0003              and #$3,d0      * sicherheitshalber
399 00000334 2F08                  move.l a0,-(a7)         * unabh. von gewaehltem Format
400 00000336 41F900000354          lea dctab1,a0
401 0000033C 0C39002100000004      cmp.b #$21,drvcode      * wenn aber MINI geladen
402 00000344 6606                  bne.s dlcode
403 00000346 41F900000358          lea dctab2,a0           * dann andere Tabelle
404                               dlcode:
405 0000034C 18300800              move.b 0(a0,d0.l),d4    * Ergebnis in D4
406 00000350 205F                  movea.l (a7)+,a0
407 00000352 4E75                  rts
408
409
410                               dctab1:         * MAXI BOOT erfolgte
411 00000354 11129192              dc.b $11,$12,$91,$92
412
413                               dctab2:         * MINI BOOT erfolge
414 00000358 14189498              dc.b $14,$18,$94,$98
415
416
417
418                               read:
419 0000035C 61000352              bsr convdrv     * 0..3 = maxi, 4..5 = min, constant
420 00000360 B03C0004              cmp.b #4,d0
421 00000364 6C74                  bge readmini
422 00000366 610001F8              bsr delsso      * maxi einstellen ggf.
```

zu Abb. 6.4.10

```
 423 0000036A 7201                  move #1,d1      * Lesen
 424 0000036C 4242                  clr d2
 425 0000036E 4243                  clr d3
 426 00000370 14390000000D          move.b sector,d2
 427 00000376 16390000000C          move.b track,d3
 428 0000037C 61AE                  bsr dcode
 429 0000037E 207900000008          movea.l dma,a0
 430 00000384 7E4B                  move #75,d7
 431 00000386 61000364              bsr trapexe
 432 0000038A 6706                  beq.s rdok
 433 0000038C 303C0001              move.w #1,d0
 434 00000390 4E75                  rts
 435                               rdok:
 436 00000392 4240                  clr.w d0
 437 00000394 4E75                  rts
 438
 439
 440
```

```
C P / M   6 8 0 0 0   A s s e m b l e r                Revision 04.03              Page   9
Source File: ndrbios.s

 441                               write:          * d1= 0=norm, 1= dirwrt, 2= first
 442 00000396 13C100000013          move.b d1,alloc
 443 0000039C 61000312              bsr convdrv     * 0..3=maxi, 4..5=mini, 6=ramflo
 444 000003A0 B03C0004              cmp.b #4,d0
 445 000003A4 6C0000B0              bge writemini
 446 000003A8 610001B6              bsr delsso
 447 000003AC 7202                  move #2,d1      * Schreiben
 448 000003AE 4242                  clr d2
 449 000003B0 4243                  clr d3
 450 000003B2 14390000000D          move.b sector,d2
 451 000003B8 16390000000C          move.b track,d3
 452 000003BE 6100FF6C              bsr dcode
 453 000003C2 207900000008          movea.l dma,a0
 454 000003C8 7E4B                  move #75,d7
 455 000003CA 61000320              bsr trapexe
 456 000003CE 6706                  beq.s wrok
 457 000003D0 303C0001              move.w #1,d0
 458 000003D4 4E75                  rts
 459                               wrok:
 460 000003D6 4240                  clr.w d0
 461 000003D8 4E75                  rts
 462
 463
 464                               readmini:
 465 000003DA B03C0006              cmp.b #6,d0
 466 000003DE 67000208              beq readram
 467 000003E2 6100010A              bsr calc
 468 000003E6 103900000010          move.b mdrvakt,d0
 469 000003EC B800                  cmp.b d0,d4     * Laufwerk gleich
 470 000003EE 6640                  bne rload       * nein, dann laden
 471 000003F0 103900000011          move.b mtrkakt,d0
 472 000003F6 B600                  cmp.b d0,d3
 473 000003F8 6636                  bne rload
 474 000003FA 103900000012          move.b msekakt,d0
 475 00000400 B400                  cmp.b d0,d2                          zu Abb. 6.4.10
 476 00000402 662C                  bne rload
 477                               rlrd:
 478 00000404 43F900000244          lea buffer,a1   * Quelle, adr. berechnen
 479 0000040A 4281                  clr.l d1
 480 0000040C 12390000000D          move.b sector,d1  * 1..40
```

```
481 00000412 5301                sub.b #1,d1     * 0..39
482 00000414 C27C0007            and #$7,d1      * 0..7
483 00000418 EF81                asl.l #7,d1     * *  128
484 0000041A D3C1                adda.l d1,a1    * ist Quelle
485 0000041C 207900000008        movea.l dma,a0 * Ziel
486 00000422 363C007F            move #128-1,d3
487                             rllp1:
488 00000426 10D9                move.b (a1)+,(a0)+
489 00000428 51CBFFFC            dbra d3,rllp1
490 0000042C 4240                clr d0
491 0000042E 4E75                rts
492
493                             rload:
494 00000430 6100017C            bsr puttrk      * falls alte Spur da, rueckschreiben
495 00000434 651C                bcs errflo

C P / M   6 8 0 0 0   A s s e m b l e r                Revision 04.03                Page  10
Source File: ndrbios.s

496 00000436 610000B6            bsr calc
497 0000043A 13C400000010        move.b d4,mdrvakt
498 00000440 13C300000011        move.b d3,mtrkakt
499 00000446 13C200000012        move.b d2,msekakt
500 0000044C 61000132            bsr gettrk
501 00000450 64B2                bcc rlrd        * und dann transfer
502                             errflo:          * GLOBAL
503 00000452 7001                move #1,d0
504 00000454 4E75                rts
505
506
507
508
509                             writemini:
510 00000456 B03C0006            cmp.b #6,d0
511 0000045A 670001C2            beq writeram
512 0000045E 6100008E            bsr calc
513 00000462 103900000010        move.b mdrvakt,d0
514 00000468 B800                cmp.b d0,d4     * Laufwerk gleich
515 0000046A 665E                bne wload       * nein, dann laden
516 0000046C 103900000011        move.b mtrkakt,d0
517 00000472 B600                cmp.b d0,d3
518 00000474 6654                bne wload
519 00000476 103900000012        move.b msekakt,d0
520 0000047C B400                cmp.b d0,d2
521 0000047E 664A                bne wload
522                             wlwr:
523 00000480 43F900000244        lea buffer,a1   * Ziel adr. berechnen
524 00000486 4281                clr.l d1
525 00000488 123900000000D       move.b sector,d1 * 1..40
526 0000048E 5301                sub.b #1,d1     * 0..39
527 00000490 C27C0007            and #$7,d1      * 0..7
528 00000494 EF81                asl.l #7,d1     * *  128
529 00000496 D3C1                adda.l d1,a1    * ist Ziel
530 00000498 207900000008        movea.l dma,a0 * Quelle
531 0000049E 363C007F            move #128-1,d3
532                             wrllp1:
533 000004A2 12D8                move.b (a0)+,(a1)+
534 000004A4 51CBFFFC            dbra d3,wrllp1
535 000004AB 13FC000100000014    move.b #1,mwrtflg * merken, das Sektor geschrieben
536 000004B0 0C3900010000013     cmp.b #1,alloc
537 000004BB 660C                bne wr2
538 000004BA 610000F2            bsr puttrk
```

zu Abb. 6.4.10

```
 539 000004BE 6692                  bne errflo
 540 000004C0 423900000010          clr.b mdrvakt   * ungueltiger Buffer
 541                                wr2:
 542 000004C6 4240                  clr d0
 543 000004C8 4E75                  rts
 544
 545
 546                                wload:
 547 000004CA 610000E2              bsr puttrk      * falls alte Spur da, rueckschreiben
 548 000004CE 6582                  bcs errflo
 549 000004D0 611C                  bsr calc
 550 000004D2 13C400000010          move.b d4,mdrvakt


C P / M   6 8 0 0 0   A s s e m b l e r              Revision 04.03              Page  11
Source File: ndrbios.s

 551 000004D8 13C300000011          move.b d3,mtrkakt
 552 000004DE 13C200000012          move.b d2,msekakt
 553 000004E4 6100009A              bsr gettrk
 554 000004E8 6500FF68              bcs errflo      * und dann transfer
 555 000004EC 6092                  bra wlwr        * und weiter dann
 556
 557
 558
 559
 560                                * Buffer Verwaltung
 561                                * fuer 1K Sektoren
 562
 563                                calc:           * Berechnet Code, LW, SEK, TRK...
 564                                *                 SEK->D2 TRK->D3 LW->D4
 565 000004EE 610001C0              bsr convdrv     * 4,5 ist immer code unabh. von Boot
 566 000004F2 0C3900210000000004    cmp.b #$21,drvcode * pruefen ob Mini-Boot.
 567 000004FA 670E                  beq calmini
 568                                *                 transfer     E,F  ist nr
 569 000004FC 183C0024              move.b #$24,d4 * LW
 570 00000500 5900                  sub.b #4,d0    * 0,1 Ergebnis
 571 00000502 6704                  beq calc1
 572 00000504 183C0028              move.b #$28,d4 * LW
 573                                calc1:
 574 00000508 600C                  bra calweiter
 575                                calmini:
 576 0000050A 183C0021              move.b #$21,d4 * dann A,B physikalisches Laufwerk
 577 0000050E 5900                  sub.b #4,d0
 578 00000510 6704                  beq calweiter
 579 00000512 183C0022              move.b #$22,d4
 580                                calweiter:
 581 00000516 143900000000D         move.b sector,d2 * 1..40
 582 0000051C 5302                  sub.b #1,d2     * 0..39
 583 0000051E E61A                  ror.b #3,d2     * nnn = 0..4
 584 00000520 C47C0007              and  #%111,d2   * Bereich Sektor          zu Abb. 6.4.10
 585 00000524 5242                  add #1,d2       * 1..5 erlaubt
 586 00000526 163900000000C         move.b track,d3
 587 0000052C E21B                  ror.b #1,d3
 588 0000052E 6404                  bcc.s calc2     * Vorderseite
 589 00000530 887C0080              or #$80,d4      * Rueckseite ansprechen// Achtung SSO verwend
en.
 590                                calc2:
 591 00000534 C67C007F              and #$7f,d3     * Track gueltig
 592 00000538 4E75                  rts
 593
 594                                *                 MINIFLOPPY
 595                                selsso:         * sso setzen, falls noetig. Und Steprate
```

```
596 000005J8 0C3900800000000F       cmp.b #$80,lastsso
597 00000542 671A                   beq nosels
598 00000544 4241                   clr d1          * step sel cmd
599 00000546 163900000001           move.b stepmini,d3
600 0000054C B67C0080               or #$80,d3
601 00000550 7E4B                   move #75,d7
602 00000552 61000198               bsr trapexe
603 00000556 13FC008000000000F      move.b #$80,lastsso
604                                 nosels:
605 0000055E 4E75                   rts


C P / M   6 8 0 0 0   A s s e m b l e r              Revision 04.03              Page 12
Source File: ndrbios.s

606
607
608                                 *                   MAXIFLOPPY
609                                 delsso:         * sso ruecksetzen. Und Steprate
610 00000560 0C3900000000000F       cmp.b #$0,lastsso
611 00000568 67F4                   beq nosels
612 0000056A 4241                   clr d1
613 0000056C 163900000000           move.b stepmaxi,d3
614 00000572 7E4B                   move #75,d7
615 00000574 61000176               bsr trapexe
616 00000578 423900000000F          clr.b lastsso
617 0000057E 4E75                   rts
618
619
620
621                                 gettrk:
622 00000580 61B8                   bsr selsso
623 00000582 423900000014           clr.b mwrtflg
624 00000588 41F900000244           lea buffer,a0  * Ziel
625 0000058E 4242                   clr d2
626 00000590 4243                   clr d3
627 00000592 143900000012           move.b msekakt,d2
628 00000598 163900000011           move.b mtrkakt,d3
629 0000059E 1B3900000010           move.b mdrvakt,d4  * drive code valid
630 000005A4 7201                   move #1,d1      * LESEN
631 000005A6 7E4B                   move #75,d7
632 000005A8 61000142               bsr trapexe     * Floppy zugriff
633 000005AC 4E75                   rts             * incl. Fehlercode
634
635
636
637                                 puttrk:
638 000005AE 4A3900000014           tst.b mwrtflg
639 000005B4 672E                   beq noput
640 000005B6 61B2                   bsr selsso
641 000005B8 423900000014           clr.b mwrtflg
642 000005BE 41F900000244           lea buffer,a0  * Quelle
643 000005C4 4242                   clr d2
644 000005C6 4243                   clr d3                                  zu Abb. 6.4.10
645 000005C8 143900000012           move.b msekakt,d2
646 000005CE 163900000011           move.b mtrkakt,d3
647 000005D4 1B3900000010           move.b mdrvakt,d4  * drive code valid
648 000005DA 7202                   move #2,d1      * SCHREIBEN
649 000005DC 7E4B                   move #75,d7
650 000005DE 6100010C               bsr trapexe     * Floppy zugriff
651 000005E2 4E75                   rts             * incl. Fehlercode
652
653                                 noput:
```

```
 654 000005E4 4240                clr d0
 655 000005E6 4E75                rts             * nicht ablegt, no error
 656
 657
 658                             readram:         * RAM FLOPPY   STARTET ab $80000 max 256K
 659 000005E8 43F900080000        lea ramflo,a1   * Quelle
 660 000005EE 4281                clr.l d1        * savety

C P / M   6 8 0 0 0   A s s e m b l e r                Revision 04.03                Page  13
Source File: ndrbios.s

 661 000005F0 12390000000C        move.b track,d1
 662 000005F6 E141                asl.w #8,d1     * 0..n
 663 000005F8 12390000000D        move.b sector,d1 * 1..256(0) ueberlauf reverse
 664 000005FE 5301                sub.b #1,d1     * 0..255
 665 00000600 EF81                asl.l #7,d1     * * 128 (da Blockgroesse)
 666 00000602 C2BC0003FFFF        and.l #$3ffff,d1 * max
 667 00000608 D3C1                adda.l d1,a1    * Adresse fertig
 668 0000060A 207900000008        movea.l dma,a0 * Ziel
 669 00000610 363C007F            move #128-1,d3
 670                             ramget:
 671 00000614 10D9                move.b (a1)+,(a0)+
 672 00000616 51CBFFFC            dbra d3,ramget
 673 0000061A 4240                clr.w d0        * no errors
 674 0000061C 4E75                rts
 675
 676
 677                             writeram:        * RAM FLOPPY   STARTET ab $80000 max 256K
 678 0000061E 43F900080000        lea ramflo,a1   * Ziel
 679 00000624 4281                clr.l d1        * savety
 680 00000626 12390000000C        move.b track,d1
 681 0000062C E159                rol.w #8,d1     * 0..n
 682 0000062E 12390000000D        move.b sector,d1  * 1..256(0) Ueberlauf reverse
 683 00000634 5301                sub.b #1,d1     * 0..255
 684 00000636 EF81                asl.l #7,d1     * * 128 (da Blockgroesse)
 685 00000638 C2BC0003FFFF        and.l #$3ffff,d1
 686 0000063E D3C1                adda.l d1,a1    * Adresse fertig
 687 00000640 207900000008        movea.l dma,a0 * Quelle
 688 00000646 4211                clr.b (a1)      * Speichertest kurz
 689 00000648 4A11                tst.b (a1)
 690 0000064A 6618                bne ramerr
 691 0000064C 12BC00FF            move.b #$ff,(a1)
 692 00000650 0C1100FF            cmp.b #$ff,(a1)
 693 00000654 660E                bne ramerr
 694 00000656 363C007F            move #128-1,d3
 695                             ramput:
 696 0000065A 12D8                move.b (a0)+,(a1)+
 697 0000065C 51CBFFFC            dbra d3,ramput
 698 00000660 4240                clr.w d0        * no errors
 699 00000662 4E75                rts
 700
 701                             ramerr:          * trotzdem copieren
 702 00000664 363C007F            move #128-1,d3
 703                             ramlput:
 704 00000668 12D8                move.b (a0)+,(a1)+
 705 0000066A 51CBFFFC            dbra d3,ramlput
 706 0000066E 7001                move #1,d0
 707 00000670 4E75                rts                                          zu Abb. 6.4.10
 708
 709
 710                             flush:
 711 00000672 6100FF3A            bsr puttrk      * incl. flags, ggf zurueckschreiben
 712 00000676 660A                bne flush1
```

```
 713 00000678 423900000010          clr.b mdrvakt  * Buffer ist dann ungueltig
 714 0000067E 4240                  clr d0         * ok
 715 00000680 4E75                  rts


C P / M   6 8 0 0 0   A s s e m b l e r                Revision 04.03              Page 14
Source File: ndrbios.s

 716                                flush1:
 717 00000682 303CFFFF              move #$ffff,d0
 718 00000686 4E75                  rts
 719
 720                                getseg:
 721 00000688 203C0000001E          move.l #memrgn,d0
 722 0000068E 4E75                  rts
 723
 724                                getiob:
 725 00000690 303900000002          move iobyte,d0
 726 00000696 4E75                  rts
 727
 728                                setiob:
 729 00000698 33C100000002          move d1,iobyte
 730 0000069E 4E75                  rts
 731                                                                  zu Abb. 6.4.10
 732                                setexc:
 733 000006A0 0281000000FF          andi.l #$ff,d1
 734 000006A6 E549                  lsl #2,d1
 735 000006A8 2041                  movea.l d1,a0
 736 000006AA 2010                  move.l (a0),d0
 737 000006AC 2082                  move.l d2,(a0)
 738                                noset:
 739 000006AE 4E75                  rts
 740
 741                                *
 742                                convdrv:        * ergibt logisches Laufwerk in D0
 743                                *                 abhaengig von drvcode
 744 000006B0 0C39002100000004     cmp.b #$21,drvcode     * ok Mini
 745 000006B8 6712                  beq.s convldrv         * dann dorthin
 746 000006BA 0C39001100000004     cmp.b #$11,drvcode     * Maxi
 747                                *                        hier Fehlerbehandlung moeglich
 748 000006C2 4280                  clr.l d0               * Langwort gueltig
 749 000006C4 10390000000E          move.b seldrv,d0       * Laufwerkscode holen 0,1,2,3,4,5,6
 750 000006CA 4E75                  rts                    * sonst direkt uebernehmen
 751                                convldrv:              * 0,1,2,3 -> 4,5 und 4,5 -> 0,1
 752 000006CC 4280                  clr.l d0
 753 000006CE 10390000000E          move.b seldrv,d0
 754 000006D4 2F08                  move.l a0,-(a7)
 755 000006D6 41F9000006E4          lea convtab,a0
 756 000006DC 10300800              move.b 0(a0,d0.l),d0   * und Wert holen, Langwort gueltig
 757 000006E0 205F                  movea.l (a7)+,a0
 758 000006E2 4E75                  rts                    * d0, gueltig
 759
 760                                convtab:
 761 000006E4 04050001020306        dc.b 4,5,0,1,2,3,6     * Konfiguration MINI-Boot.
 762 000006EC                       ds 0
 762 000006EC
 763
 764
 765                                trapexe:        * Grundprog ausfuehren ueber Sprung
 766 000006EC 48E70006              movem.l a5/a6,-(a7)
 767 000006F0 2C790000001A          move.l gruj68k,a6 * Zieladresse auf Stack
 768 000006F6 4E96                  jsr (a6)          * Sprung auf TRAP-Ersatz
 769 000006F8 4CDF6000              movem.l (a7)+,a5/a6
```

```
C P / M   6 8 0 0 0   A s s e m b l e r                Revision 04.03              Page  15
Source File: ndrbios.s

 770 000006FC 4E75                    rts
 771                                 *
 772
 773 00000000                         .data
 774
 775
 776 00000000 00                     stepmaxi: dc.b 0 * 0=schnellste ,1,2,3
 777 00000001 00                     stepmini: dc.b 0 * 0=schnellste ,1,2,3
 778 00000002 0000                   iobyte:   dc.w 0 * nur Byte gueltig.
 779
 780                                 * drvcode bestimmt das gebootete Laufwerk
 781                                 * er entspricht der internen Codierung,
 782                                 * also $11 bei Maxi, SD
 783                                 * und  $21 bei Mini, DD
 784                                 * nur diese beiden Formate werden z.Z. ausgewertet
 785
 786 00000004 00                     drvcode: dc.b 0
 787 00000005 00                              dc.b 0   * angleich EVEN
 788
 789                                 **
 790 00000006 0000                   cursor:  dc.w 0   * Merker Cursor 1=ein, 0=aus
 791 00000008 00000000               dma:     dc.l 0
 792 0000000C 00                     track:   dc.b 0
 793 0000000D 00                     sector:  dc.b 0
 794
 795 0000000E 00                     seldrv:  dc.b 0
 796 0000000F FF                     lastsso: dc.b $FF * Merker letzter SSO-Zustand (0,$80) FF=ung
uelt.
 797                                 *                    0=ohne SSO, $80=SSO war gesetzt
 798 00000010 00                     mdrvakt: dc.b 0   * Intern code 0=unguelt.
 799 00000011 00                     mtrkakt: dc.b 0   * fuer Bufferverwaltung
 800 00000012 00                     msekakt: dc.b 0
 801 00000013 00                     alloc:   dc.b 0
 802
 803 00000014 00                     mwrtflg: dc.b 0   * <> 0 dann schreiben noch noetig
 804
 805 00000016                         ds 0
 805 00000016
 806
 807 00000016 00000000               gru68k:  dc.l 0         * Basis-Adresse Grundprogramm
 808 0000001A 00000420               gruj68k: dc.l $420      * Basis-Adresse + $420 fuer Sprung
 809
 810
 811                                 meargn:
 812 0000001E 0001                    dc.w 1              * nur eine TPA vorhanden (immer so.)
 813 00000020 00000400                dc.l $400           * Start der TPA
 814 00000024 FFFFFC00                dc.l cpm-$400       * 1024 Bytes gehen durch Vektoren verlor
en
 815
 816                                 * disk parameters
 817
 818                                 maxdsk equ 7
 819                                 dphlen equ 26
 820
 821                                 dph0:    * 8 "  a                          zu Abb. 6.4.10
 822 00000028 0000010E                dc.l xlt
 823 0000002C 0000                    dc.w 0
```

```
C P / M   6 8 0 0 0   A s s e m b l e r                Revision 04.03          Page  16
Source File: ndrbios.s

 824 0000002E 0000                  dc.w 0
 825 00000030 0000                  dc.w 0
 826 00000032 00000000              dc.l dirbuf
 827 00000036 000000DE              dc.l dpb
 828 0000003A 000000B0              dc.l ckv0
 829 0000003E 00000140              dc.l alv0
 830
 831                               dph1:    * 8 " b
 832 00000042 0000010E              dc.l xlt
 833 00000046 0000                  dc.w 0
 834 00000048 0000                  dc.w 0
 835 0000004A 0000                  dc.w 0
 836 0000004C 00000000              dc.l dirbuf
 837 00000050 000000DE              dc.l dpb
 838 00000054 00000090              dc.l ckv1
 839 00000058 00000160              dc.l alv1
 840
 841                               dph2:    * 8 " c
 842 0000005C 0000010E              dc.l xlt
 843 00000060 0000                  dc.w 0
 844 00000062 0000                  dc.w 0
 845 00000064 0000                  dc.w 0
 846 00000066 00000000              dc.l dirbuf
 847 0000006A 000000DE              dc.l dpb
 848 0000006E 000000A0              dc.l ckv2
 849 00000072 000001B0              dc.l alv2
 850
 851                               dph3:    * 8 " d
 852 00000076 0000010E              dc.l xlt
 853 0000007A 0000                  dc.w 0
 854 0000007C 0000                  dc.w 0
 855 0000007E 0000                  dc.w 0
 856 00000080 00000000              dc.l dirbuf
 857 00000084 000000DE              dc.l dpb
 858 00000088 000000B0              dc.l ckv3
 859 0000008C 000001A0              dc.l alv3
 860
 861                               dph4:  * 5 1/4 " e
 862 00000090 00000000              dc.l 0          * no table
 863 00000094 0000                  dc.w 0
 864 00000096 0000                  dc.w 0
 865 00000098 0000                  dc.w 0
 866 0000009A 00000000              dc.l dirbuf
 867 0000009E 000000EE              dc.l dpbmini
 868 000000A2 000000C0              dc.l ckv4
 869 000000A6 000001C0              dc.l alv4
 870
 871                                                                  zu Abb. 6.4.10
 872                               dph5:    * 5 1/4 " f
 873 000000AA 00000000              dc.l 0  * no table
 874 000000AE 0000                  dc.w 0
 875 000000B0 0000                  dc.w 0
 876 000000B2 0000                  dc.w 0
 877 000000B4 00000000              dc.l dirbuf
 878 000000B8 000000EE              dc.l dpbmini
```

```
C P / M   6 8 0 0 0   A s s e m b l e r                Revision 04.03              Page 17
Source File: ndrbios.s

 879 000000BC 00000100              dc.l ckv5
 880 000000C0 000001F2              dc.l alv5
 881
 882
 883                               dph6:    * Ramfloppy g
 884 000000C4 00000000              dc.l 0  * no table
 885 000000C8 0000                  dc.w 0
 886 000000CA 0000                  dc.w 0
 887 000000CC 0000                  dc.w 0
 888 000000CE 00000000              dc.l dirbuf
 889 000000D2 000000FE              dc.l dpbram
 890 000000D6 00000000              dc.l 0  * no check
 891 000000DA 00000224              dc.l alv6
 892
 893
 894                               dpb:
 895 000000DE 001A                  dc.w 26
 896 000000E0 03                    dc.b 3
 897 000000E1 07                    dc.b 7
 898 000000E2 00                    dc.b 0
 899 000000E3 00                    dc.b 0
 900 000000E4 00F2                  dc.w 242
 901 000000E6 003F                  dc.w 63
 902 000000E8 C000                  dc.w $c000
 903 000000EA 0010                  dc.w 16
 904 000000EC 0002                  dc.w 2
 905
 906                               dpbmini:
 907 000000EE 0028                  dc.w 40         * 40 log. 128 Byte Sektoren
 908 000000F0 04                    dc.b 4          * BSH    2048 Bytes / Block
 909 000000F1 0F                    dc.b 15         * BLM
 910 000000F2 00                    dc.b 0          * EXTNT MASK
 911 000000F3 00                    dc.b 0
 912 000000F4 0184                  dc.w 388        * Kapazitaet * 2048
 913 000000F6 00FF                  dc.w 255        * Direktory Eintraege-1
 914 000000F8 F000                  dc.w $f000      * Direktory mask (reserved )
 915 000000FA 0040                  dc.w 64         * check size
 916 000000FC 0004                  dc.w 4          * offset
 917
 918
 919                               dpbram:
 920 000000FE 0100                  dc.w 256        * 256 log. 128 Byte Sektoren
 921 00000100 03                    dc.b 3          * BSH    1024 Bytes / Block
 922 00000101 07                    dc.b 7          * BLM
 923 00000102 00                    dc.b 0          * EXTNT MASK
 924 00000103 00                    dc.b 0
 925 00000104 00FF                  dc.w 255        * Kapazitaet * 1024 , max 256K patch
 926 00000106 003F                  dc.w 63         * Direktory Eintraege-1
 927 00000108 C000                  dc.w $c000      * Direktory mask (reserved )
 928 0000010A 0000                  dc.w 0          * no check
 929 0000010C 0000                  dc.w 0          * offset
 930
 931                               xlt:
 932 0000010E 01070D13              dc.b 1,7,13,19
 933 00000112 19050B11              dc.b 25,5,11,17
```

zu Abb. 6.4.10

```
C P / M   6 8 0 0 0   A s s e m b l e r                Revision 04.03                Page  18
Source File: ndrbios.s

 934 00000116 1703090F              dc.b 23,3,9,15
 935 0000011A 1502080E              dc.b 21,2,8,14
 936 0000011E 141A060C              dc.b 20,26,6,12
 937 00000122 1218040A              dc.b 18,24,4,10
 938 00000126 1016                  dc.b 16,22
 939
 940 00000128                       ds 0
 941
 942 00000000                       .bss
 943
 944                               dirbuf:
 945 00000000                       ds.b 128
 946 00000080                      ckv0: ds.b 16
 947 00000090                      ckv1: ds.b 16
 948 000000A0                      ckv2: ds.b 16
 949 000000B0                      ckv3: ds.b 16
 950 000000C0                      ckv4: ds.b 64
 951 00000100                      ckv5: ds.b 64
 952
 953 00000140                      alv0: ds.b 32
 954 00000160                      alv1: ds.b 32
 955 00000180                      alv2: ds.b 32
 956 000001A0                      alv3: ds.b 32
 957 000001C0                      alv4: ds.b 50          * (49+1), Achtung, alvs auf even
 958 000001F2                      alv5: ds.b 50          * sonst Fehler in STAT
 959 00000224                      alv6: ds.b 32
 960
 961
 962 00000244                      buffer: ds.b 1024      * Sektor Speicher fuer Mini-Laufwerk
 963
 964
 965 00000644                       ds 0
 966
 967 00000644                      .end
```

Abb. 6.4.10 Das vollständige BIOS des CP/M-68k

Programm CPM256.SYS, dies ist das CP/M für den Betrieb mit 256 KByte RAM. Wenn Sie soviel RAM haben, können Sie das neue CP/M einfach dadurch installieren, daß Sie das alte z.B. in CPM128.SYS umbenennen und das neue in CPM.SYS. Beim nächsten Kaltstart vom Grundprogramm-Menue aus wird es dann automatisch verwendet. Jede andere Speicherkonfiguration kann auch erzeugt werden; wie bereits gesagt, brauchen Sie dazu das Handbuch. Nur ein Hinweis, Sie müssen keine Änderung am BIOS vornehmen, sondern nur die Angabe beim „RELOC" entsprechend durchführen, nachdem alle Programme übersetzt und gebunden wurden.

Das BIOS ist sowohl für 5¼"-Disketten als auch für 8"-Disketten geeignet. Außerdem stellt es uns eine RAM-Floppy zur Verfügung.

Dazu noch ein paar Hinweise:
Wenn man das System auf 8"-Disketten startet, so ergibt sich folgende Laufwerks-Zuweisung:
Laufwerk A = Vorderseite des 8"-Laufwerks mit DS0
Laufwerk B = Vorderseite des 8"-Laufwerks mit DS1
Laufwerk C = Rückseite des 8"-Laufwerks mit DS0
Laufwerk D = Rückseite des 8"-Laufwerks mit DS1

Die Rückseiten müssen mit UFORM68K (ebenfalls auf der Diskette) ohne SSO formatiert werden, also ohne gesetztes Seiten-Auswahl-Bit. Ferner gilt:
Laufwerk E = 5¼"-Laufwerk (Vorder- und Rückseite) DS2
Laufwerk F = 5¼"-Laufwerk (Vorder- und Rückseite) DS3
Laufwerk G = Ramfloppy, 256 KByte, ab Adresse $80000.

Bei einem Schreibversuch mit nicht vorhandener Ramfloppy wird „BAD SECTOR" gemeldet.

Wenn man vom 5¼"-Laufwerk aus bootet, sind Laufwerke A und B die 5¼"-Laufwerke (DS0 und DS1) und C,D,E,F die 8"-Laufwerke.

Jetzt gibt es noch eine interessante Eigenschaft. Das CP/M-68k besitzt vier sogenannte logische Kanäle, die mit CON:, AUXI:, AUXO: und LIST: benannt sind. CON: ist normalerweise die Konsole, also unsere Tastatur und der Bildschirm, der mit der GDP64-Baugruppe realisiert ist. LIST: ist normalerweise der Centronics-Port mit der IOE-Baugruppe (siehe Kapitel 7), und AUXI: und AUXO: bedienen die serielle Schnittstelle, die übrigens mit 9600 Baud programmiert wird; eine Änderung ist durch ein Hilfsprogramm (siehe SIIINIT im Grundprogramm) leicht möglich. Man kann nun diese logischen Kanäle auch anderen physikalischen Geräten zuordnen. Also die Konsole auf die serielle Schnittstelle schalten und dann über ein getrenntes Datensichtgerät bedienen! Dazu gibt man z.B. den Befehl:
STAT CON:=CRT:

Alle Kommandos werden nun von der seriellen Schnittstelle in Empfang genommen und alle Ausgabe werden dorthin gesendet. So kann man z.B. ein schnelles Datensichtgerät anschließen, wenn die GDP64 für eine Textverarbeitung zu langsam ist.

Ferner kann man auch die Druckerausgabe auf die serielle Schnittstelle umlenken:
STAT LST:=CRT:

Im CP/M-68k ist unter dem STAT-Befehl alles beschrieben, im BIOS-Listing finden Sie Hinweise, wie die einzelnen physikalischen Geräte belegt sind.

Durch die Eingabe des Kommandos „STAT DEV:" kann man sich die aktuelle Belegung anzeigen lassen, „STAT VAL:" gibt alle Möglichkeiten für STAT aus.

Hilfsprogramme für das Arbeiten mit CP/M-68k

Auf der Diskette CP/M-68k befindet sich auch ein Programm mit dem Namen „ED.68K". Dies ist der mitgelieferte Editor, er ist jedoch nur zeilenorientiert. Da wir im Grundprogramm über einen bildschirmorientierten Editor vefügen, liegt der Gedanke nahe, beides zu kombinieren. Mit einem zeilenorientierten Editor läßt sich immer nur eine Zeile bearbeiten. Dafür ist er bei großen Dateien schneller als unser Bildschirm-Editor. Auf der Systemdiskette befindet sich ein Programm, das es gestattet, unseren Bildschirmeditor unter CP/M zu verwenden. Das Programm wird mit dem CP/M-68k-Assembler übersetzt und als EDITRDK.68K Datei gebunden. Wie das genau abläuft, ist ebenfalls der CP/M-68k Literatur zu entnehmen. Nach dieser Bearbeitung kann man einfach Programme editieren, indem man unter CP/M eingibt:

EDITRDK name.ext

Falls in der angegebenen Datei bereits ein Text vorhanden war, so wird dieser automatisch geladen. Anschließend wird der Editor des Grundprogramms aufgerufen. Wenn man den Editorvorgang mit CTRL-K und „X" beendet, so wird das Programm wieder auf die Diskette zurückgeschrieben, und man landet wieder im CP/M-Betriebssystem.

Alle Grundprogrammfunktionen können mit dem Mechanismus im BIOS aufgerufen werden. Dazu enthält unser BIOS einen weiteren TRAP-Eingang mit dem Index 23. Damit ist es möglich, die Startadresse des Grundprogramms festzustellen. Im Grundprogramm steht ab der Adresse $400 vom Anfang entfernt eine Sprungtabelle mit Sprüngen zu den einzelnen Funktionen. Den TRAP-Aufruf TRAP 1 kann man normalerweise unter CP/M nicht verwenden, da er vom Betriebssystem überschrieben wird. Man könnte ihn natürlich wieder zurücksetzen und dann wie gewohnt arbeiten.

Ein weiteres nützliches Programm ist ebenfalls auf der Diskette. Damit kann man auf der Diskette gespeicherte Programme mit dem im Grundprogramm eingebauten Assembler übersetzen. Das geht wesentlich schneller als mit dem CP/M-68k-Assembler, der über 50 K lang ist.

Durch das Programm „STARTE" läßt sich ein so übersetztes Programm dann starten. Beispiel:

STARTE START

Wenn Sie Sprachen wie C, Pascal, Forth oder Basic verwenden, denken Sie daran, daß alle Grundprogramm-Funktionen immer zur Verfügung stehen. Zum einen können Sie sie über den CHR(1)-Mechanismus aufrufen (siehe Pascal/S-Beispiele im Kapitel 4) oder über den TRAP-Befehl im BIOS, siehe Beispiele wie EDITRDK etc.

Das Arbeiten unter CP/M-68k ist manchmal nicht ganz einfach, auch hier heißt es: Übung macht den Meister.

7 Die Baugruppen

In diesem Kapitel werden die Baugruppen beschrieben. Alle Baugruppen, die im Buch „Mikrocomputer selbstgebaut und programmiert" schon beschrieben wurden, werden hier bezüglich des Aufbaus nur nochmals kurz dargestellt, die neuen hinzu gekommenen erfahren dagegen eine ausführliche Erläuterung. Es gibt verschiedene Wege, um in den Besitz des erforderlichen Aufbaus zu kommen.

1. Man baut alles selbst. Dazu sind alle Schaltbilder abgedruckt. Die Bauteile kann man auch auf Lochraster-Platten aufbauen. Dieses Verfahren ist aber wirklich nur für sehr Geübte geeignet.

2. Man besorgt sich nur die Leiterplatten vom Hersteller (siehe Bezugsquellenverzeichnis im Anhang). Diese Methode ist zu empfehlen, wenn man gute Bezugsquellen von Bauteilen kennt. Die Beschaffung mancher Teile ist aber nicht immer ganz einfach.

3. Man kauft sich die Bausätze. In den Bausätzen befinden sich zusätzliche Bauanleitungen, die den Aufbau insbesondere für den Anfänger erleichtern.

4. Es werden fertige Gruppen bezogen. Dies ist der schnellste Weg, zu einem funktionsfähigen Computer zu kommen und für alle eiligen Leser zu empfehlen.

In diesem Kapitel sind alle Informationen abgedruckt, die für einen Aufbau in eigener Regie oder auch für Messungen erforderlich sind. Dabei ist insbesondere auf die Fehlersuche großer Wert gelegt worden. In den einzelnen Abschnitten finden sich Impulsdiagramme, die mit einem Logik-Analysator aufgenommen wurden. Diese Diagramme geben Anhaltspunkte für das richtige Aussehen der Impulse an den Meßpunkten. Zur Fehlersuche kann man entweder einen einfachen Logik-Prüfstift oder ein Oszilloskop verwenden.

1. Mit einem Prüfstift ist man in der Lage, an den Meßpunkten vorhandene Pulse von statischen Signalpegeln zu unterscheiden. Damit lassen sich bereits die wichtigsten Meßaufgaben lösen. Die abgedruckten Pulsdiagramme helfen bei der Orientierung.

2. Mit einem Oszilloskop kann man auch die einzelnen Pulsformen kontrollieren. Dabei ist es nützlich, ein triggerbares Skop zu besitzen; dann genügt auch ein einfaches Ein-Kanal-Gerät. Es reicht für Zeitvergleiche aus, wenn man auf ein Diagramm triggert und ein anderes darstellt.

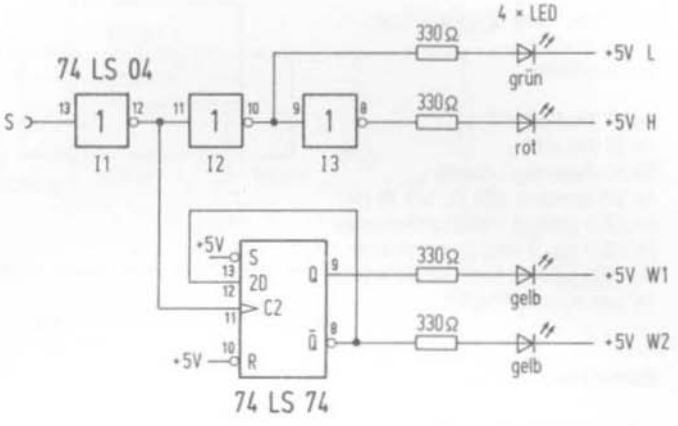


Abb. 7.1
Schaltbild des Prüfstiftes

	L L1	H L2	W1 L3	W2 L4
S = 0-Signal	✱	○	✱	○
S = 0-Signal	✱	○	○	✱
S = 1-Signal	○	✱	✱	○
S = 1-Signal	○	✱	○	✱
S = ⎍⎍	⊛	⊛	⊛	⊛
S = ⎍⎍	✱	○	⊛	⊛
S = ⎍⎍	○	✱	⊛	⊛

✱ leuchtet hell
⊛ leuchtet halb hell
○ dunkel

Abb. 7.2
Verschiedene Anzeigekombinationen

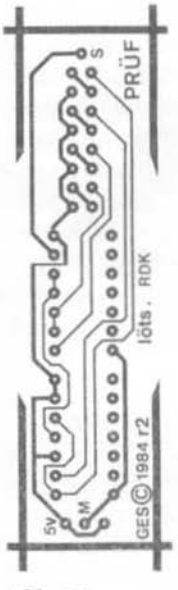


Abb. 7.3
Die Lötseite
der Leiterplatte
des Prüfstiftes

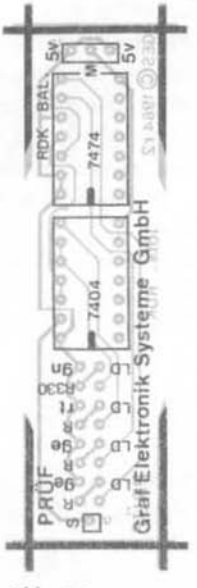


Abb. 7.4
Bestückungsplan

Aufbau des Prüfstiftes:

Materialliste:

1× IC 74LS04
1× IC 74LS74
2× IC-Fassung 14polig
4× Widerstand 330 Ω, 1/8 W (¼)
2× LED gelb, 3 mm Durchmesser
1× LED rot, 3 mm Durchmesser
1× LED grün, 3 mm Durchmesser
1× Leiterplatte PRUEF

Kenndaten:

Betriebsspannung: 5 V
Stromaufnahme: 30 mA

Abb. 7.1 zeigt das Schaltbild des Prüfstiftes. Der Eingang S ist der Eingang des Prüfstiftes; er wird an die Meßstelle gesetzt. Das auszuwertende Signal wird über drei Nicht-Glieder aufbereitet. Das erste Nicht-Glied I1 hat die Aufgabe, die restliche Schaltung vom Prüfpunkt S zu trennen. Dadurch wird die Meßstelle nicht zu stark belastet. Dann gelangt das Signal an den Takteingang eines Flipflops, an dessen Ausgänge die Leuchtdioden W1 und W2 angeschlossen sind. Das Flipflop teilt den Eingangstakt, so daß an den LEDs optisch erfaßbare Einzelimpulse ankommen. Die Leuchtdioden L und H dienen der Anzeige von „Low" und „High" also von Logik-Pegeln. Liegt ein symmetrischer Takt am Eingang, so leuchten alle Leuchtdioden auf. Es ist also möglich, verschieden Signalformen abzuschätzen, bzw. zu erkennen. *Abb. 7.2* zeigt eine Zusammenstellung von verschiedenen Kombinationen.

In *Abb. 7.3* ist das Layout, in *Abb. 7.4* der Bestückungsplan des Prüfstiftes zu sehen. Den ganzen Aufbau kann man in einem formschönen Gehäuse unterbringen.

Hinweis zum Aufbau: die Leuchtdioden müssen polrichtig eingebaut werden. Eine Fehlpolung schadet zwar nicht, läßt die Leuchtdioden aber dunkel. Wenn man herausfinden will, wo die Katode ist, so kann man die LED über einen Vorwiderstand an eine Batterie anschalten. Normalerweise ist der Katodenanschluß durch eine Abflachung am Gehäuse oder durch einen kürzeren Anschlußdraht gekennzeichnet.

7.1 Die Spannungsversorgung

Natürlich muß unser System mit einer brauchbar aufbereiteten Spannung versorgt werden. TTL-Schaltkreise und Mikrocomputer benötigen eine stabilisierte Spannung, die innerhalb bestimmter Grenzen liegen muß. So darf die 5 V-Spannung nur um 5 % abweichen. Während es bei Abweichungen nach unten nur zu Störungen kommt, können bereits kurzzeitig höhere Spannungen zur Zerstörung von Bauteilen führen.

Glücklicherweise gibt es integrierte Spannungsregler, die in der Lage sind alle Anforderungen zu erfüllen.

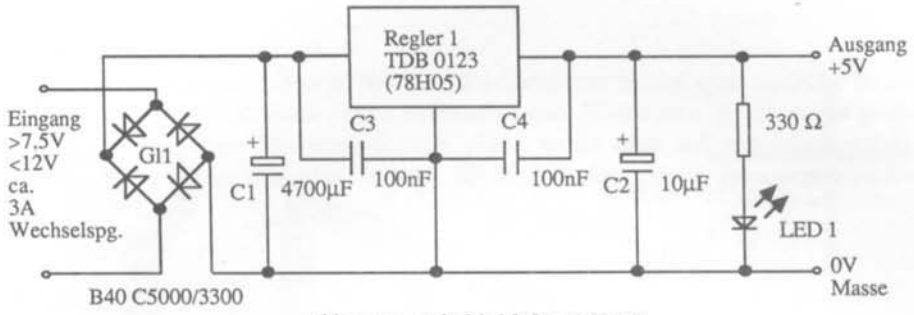


Abb. 7.1.1 Schaltbild der POW5V

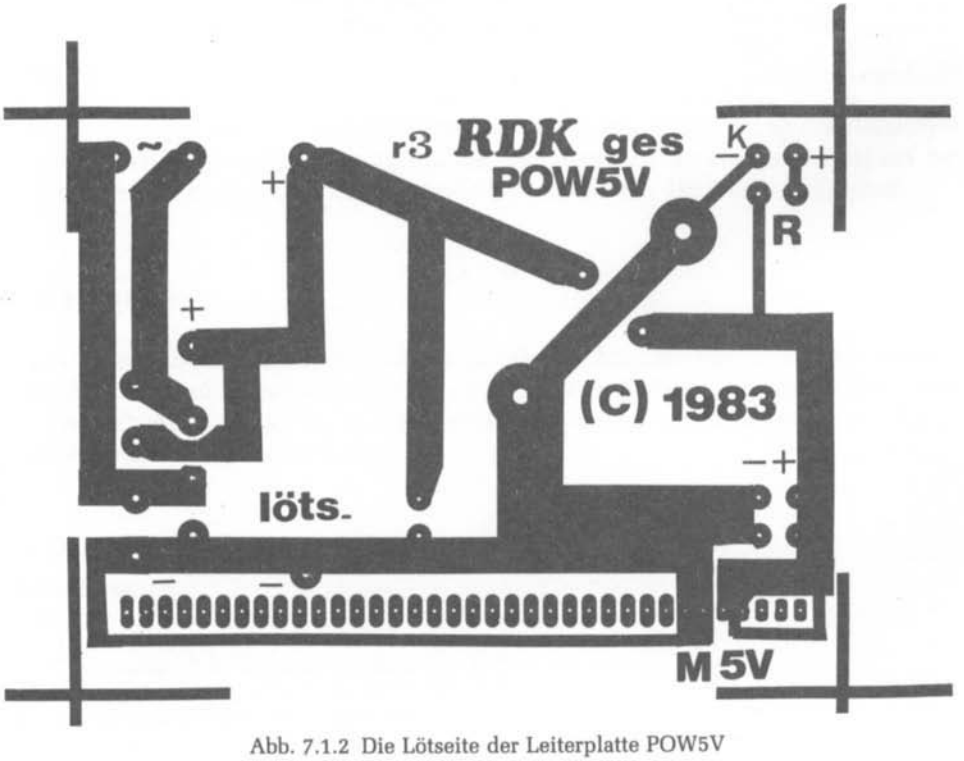


Abb. 7.1.2 Die Lötseite der Leiterplatte POW5V

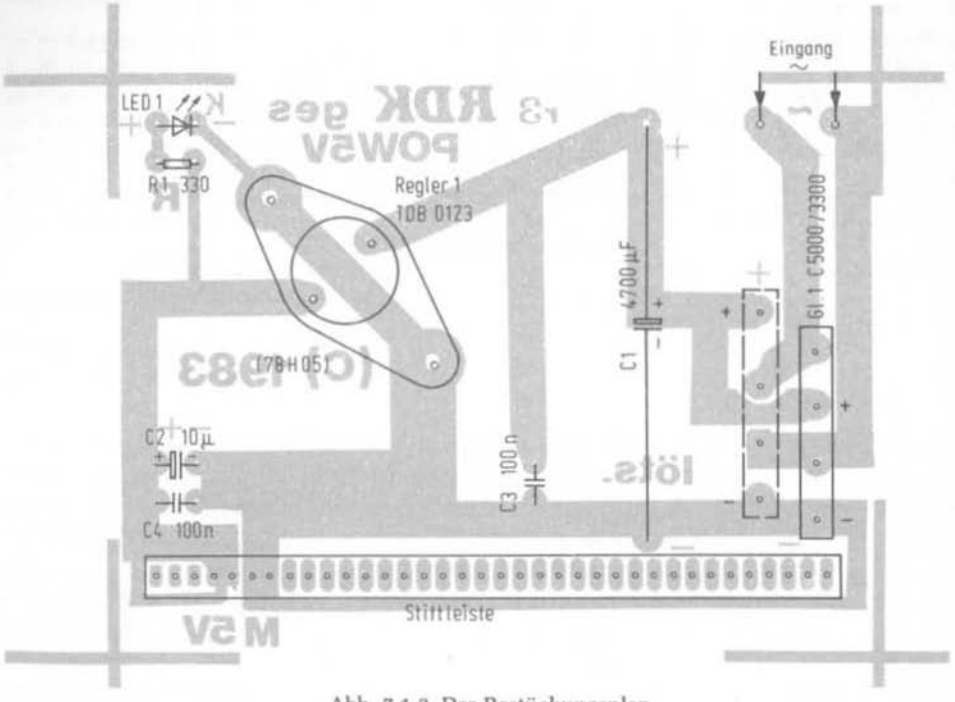


Abb. 7.1.3 Der Bestückungsplan

5-V-Versorgung

Für die ersten Versuche mit dem NDR-KLEIN-Computer genügt eine einfache Versorgungsspannung, die man auch selbst aufbauen kann. Wenn man gleich einen großen Ausbau, z.B. mit einer Disketten-Station plant, sollte man auf ein kommerzielles Netzgerät ausweichen, das eine höhere Leistung liefert (siehe Bezugsquellen-Verzeichnis).

Materialliste:

1× Gleichrichter B40 C5000/3300
1× Regler TDB 0123 (3 A) oder 78H05 (5 A)
1× 4700 µF 40 V
1× 10 µF 16 V
2× 100 nF
1× Widerstand 330 Ω
1× LED
1× Stiftleiste 36polig
1× Leiterplatte POW5V
1× Fingerkühlkörper.
1× Trafo, VDE, ca. 7.5 V bis 12 V mit 3 A

Abb. 7.1.1 zeigt den Schaltplan. Die Baugruppe wird durch eine Wechselspannung versorgt, die von einem Trafo kommt. Dieser Trafo sollte eine Spannung zwischen 7.5 V und maximal 12 V liefern. Dabei muß ein Strom von 3 A möglich sein. *Abb. 7.1.2* zeigt das Layout von der Lötseite aus und *Abb. 7.1.3* den Bestückungsplan.

Hinweis: achten Sie beim Aufbau insbesondere auf die richtige Polung des Elektrolyt-Kondensators! Die Minusmarkierung des Kondensators muß mit der Markierung auf der Leiterplatte übereinstimmen. Zur Kontrolle verwenden Sie den Bestückungsplan.

Fertig-Netzteile

Wenn Sie ein kommerzielles Netzteil anschließen, achten Sie auf die richtige Verbindung der einzelnen Spannungen mit dem BUS des NDR-KLEIN-Computers. Verwenden Sie vorsichtshalber ein Meßgerät, um vor dem Einstecken von Baugruppen alle Spannungen nochmals zu kontrollieren.

7.2 Der BUS

Materialliste:

1× Bus-Leiterplatte
36× 18polige Buchsenleiste, einreihig, Typ AMP.
oder 33× 18polige Buchsenleiste, einreihig, Typ AMP
\+ 3× 18polige Buchsenleiste, doppelreihig, Typ AMP.
2× Leiterplattenhalterung der Firma GES.

Der Bus hat die Aufgabe, die Signalleitungen der einzelnen Baugruppen miteinander zu verbinden. Dabei werden fast alle Leitungen parallel verbunden. Das Wort Bus kommt aus dem lateinischen Omnibus, für alle. Alle Baugruppen können diese Busleitungen nämlich gemeinsam benutzen.

Auch die Versorgungsspannung wird über diesen Bus an die Baugruppen geleitet. Unsere BUS-Platine kann bis zu 12 Baugruppen aufnehmen. Auf der Lötseite befinden sich die Verbindungsleitungen, auf der Bestückungsseite im wesentlichen nur eine große Massefläche. Diese Fläche ist an mehreren Durchführungsstellen mit der Masseleitung auf der Lötseite verbunden.

Beim Einlöten der Buchsenleisten kann die mittlere Buchsenreihe doppelreihig ausgeführt werden. Dies ist nötig, wenn man später mit der 16-Bit-68000-CPU mit dem 16-Bit-Datenbus arbeiten will. Sie können auch dann, wenn diese doppelreihige Leiste eingebaut ist, normale einreihige Baugruppen an diese Stelle stecken.

Alle Lötungen werden ausschließlich auf der Lötseite durchgeführt. *Abb. 7.2.1* zeigt die Belegung der Bus-Anschlüsse.

Die einzelnen Signale:

-5 V, +12 V, -12 V, +5 V, 0 V; dies sind die Versorgungs-Spannungs-Anschlüsse des Busses. Dorthin müssen die Versorgungsleitungen gelegt werden. Ein Großteil der Baugruppen kommt mit +5 V aus, jedoch gibt es welche, die auch andere Spannungen benötigen (SER, AD10x1, FLO2). Wenn Sie die POW5V-Baugruppe verwenden, so werden die 5 V- und 0 V-Anschlüsse automatisch richtig versorgt. Bei der Verwendung eines externen Netzgerätes müssen die Leitungen aus einer starken Litze bestehen.

D0...D7; dies sind die Datenleitungen. Alle Dateninformationen vom und zum Mikroprozessor werden über diese Leitungen geführt. Im 68008-System ist dieser Bus acht Bit breit; es liegen also acht Leitungen parallel. Soll der 68000 zum Einsatz kommen, so sind 16 Leitungen erforderlich. Um diese Anzahl zur Verfügung stellen zu können, wird der Bus in der Mitte geteilt. Die Leitungen D0 bis D7 und -RD, -WR, -IORQ und -MREQ werden zwischen den Reihen der doppelreihigen Buchse getrennt. Die Datenleitungen D8 bis D15 liegen dann links, D0 bis D7 rechts. Bei einem 32-Bit-Datenbus, wie er z.B. für die Prozessoren 68020 und 32032 nötig ist, werden zwei große Busbaugruppen verwendet. *Abb. 7.2.2* zeigt die verschiedenen Möglichkeiten.

-RD; damit teilt die CPU der Peripherie mit, daß sie lesen will. Das Signal ist aktiv, wenn es auf Low liegt.

-WR; damit wird eine Schreibanforderung ausgegeben. Das Signal ist aktiv, wenn es auf Low liegt.

-IORQ; eine Peripherieadresse soll angesprochen werden. Beim 68008 und 68000 wird eine 16-Bit-Adresse an A0 bis A15 als Peripherie-Adresse ausgegeben. Die hier beschriebenen Baugruppen werten allerdings nur A0 bis A7 aus. Die Datenrichtung wird durch -RD oder -WR bestimmt.

-MREQ; eine Speicheradresse wird angesprochen. A0 bis A19 geben dabei die Adresse an. -RD und -WR steuern die Datenrichtung.

A0 ... A7; die unteren (niederwertigen) Adreßleitungen.

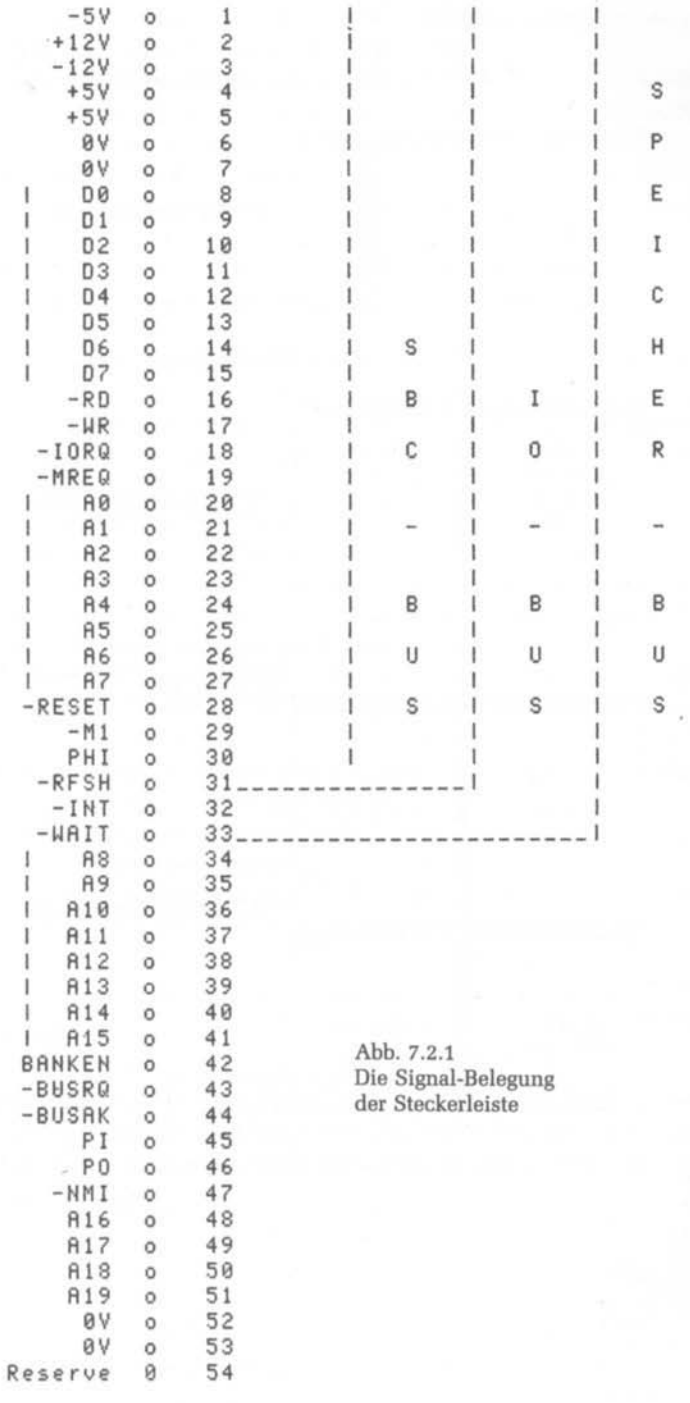


Abb. 7.2.1
Die Signal-Belegung
der Steckerleiste

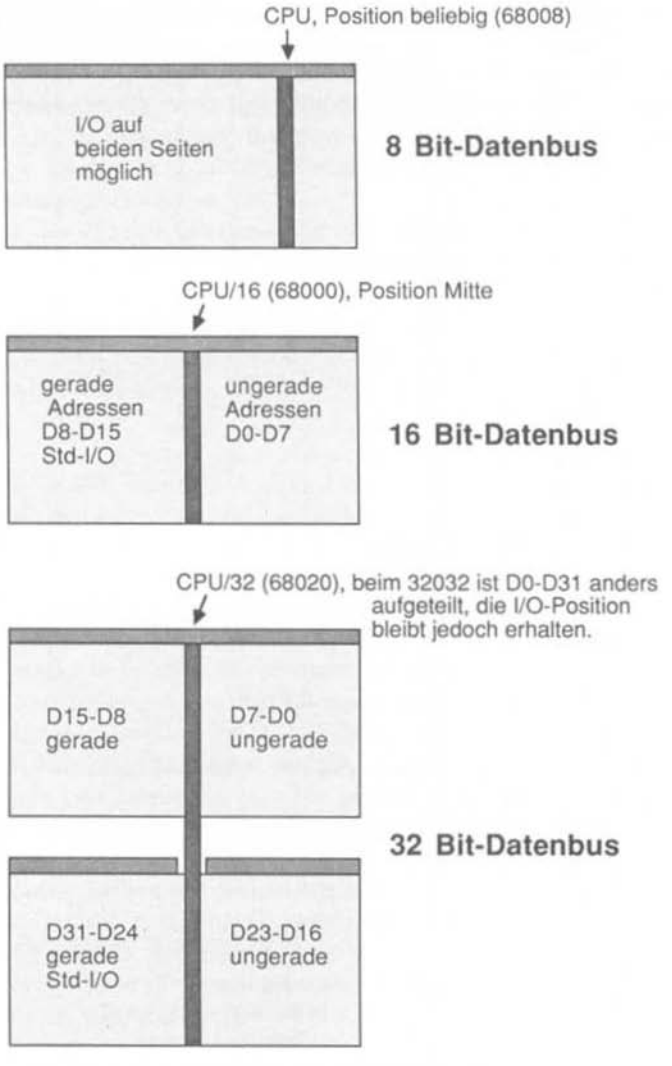


Abb. 7.2.2 Unterschiedliche Datenbus-Anordnungen

-RESET; das Signal bringt die CPU in einen definierten Zustand, es ist nach dem Einschalten oder bei einem manuellen Reset auf Low.

-M1; das Signal ist nur beim Z80 verwendet und gibt den Operations-Zyklus an. Beim 68008/68000 wird es nicht auf den Bus gelegt. Beim Z80 besitzt es noch die Bedeutung einer Interrupt-Bestätigung, wenn es zusammen mit -IORQ auftaucht.

-PHI; dies ist der Systemtakt. Er kann – je nach CPU (4MHz bis 12MHz) – unterschiedlich hoch sein und sollte daher nicht als Arbeitstakt für Peripherie-Bausteine verwendet werden.

-RFSH; diese Ausgabeleitung mancher CPUs (z.B. Z80) leitet den für dynamische RAMs notwendigen Auffrischzyklus ein. Wir verwenden es nicht unmittelbar in unseren Speichergruppen.

-INT; dies ist die Auslöseleitung für den maskierbaren Interrupt; sie ist low-aktiv.

-WAIT; damit kann man den Prozessor anhalten. Die Wartezeit sollte einige Mikrosekunden nicht überschreiten, da ggf. auch der Refresh dynamischer Speicher für diese Zeit ausfällt. -WAIT sollte nur für Synchronisationszwecke verwendet werden.

A8 .. A15; die oberen Adreßleitungen.

BANKEN; das Signal erlaubt es, Speicherbaugruppen auszublenden; es ist high-aktiv. Ist BANKEN aktiv, sind alle Speicherbaugruppen freigegeben. Liegt es auf Low, müssen sie inaktiv sein. Dieses Signal dient dem Einblenden von anderen Speicherbereichen, z.B. EPROMs beim Starten oder Graphik-Bild-Speichern für spezielle Video-Baugruppen.

-BUSRQ; damit kann eine DMA-Anforderung an die CPU gegeben werden.

-BUSAK; dies ist die Antwort der CPU auf eine durch BUSRQ eingeleitete DMA-Anforderung.

PI (Priority In); es dient üblicherweise für die Regelung in einer Interrupt-Prioritätskette und wird derzeit nicht verwendet.

PO (Priority Out); es bildet gemeinsam mit PI die Kette.

-NMI; ein nicht-maskierbarer Interrupt.

A16..A19; weitere vier Adreßleitungen, die eine Erweiterung auf 1MByte Speicher erlauben.

7.3 Die CPU68K

Das wichtigste Element des Computers ist die CPU, oder ausgeschrieben „central processing unit", zu deutsch Zentraleinheit. Sie bildet den Kern des Computers; alle Verknüpfungen, Operationen etc. laufen in ihr ab. Wir verwenden normalerweise den Mikroprozessor MC 68008.

Materialliste:

IC1 NE 555, Timer
IC2 7405, Inverter mit offenem Kollektor
IC3,IC5 74LS04, Inverter
IC4 74LS164, Schieberegister
IC6 MC 68008 L8 oder P8, Mikroprozessor
IC7 74LS74, D-Flipflops
IC8 74LS20, 4fach-Nand-Verknüpfungen
IC9 74LS245, bidirektionaler Bustreiber
IC10 74LS139, 1 aus 4 Dekoder
IC11,IC12,IC13 74LS373, 8fach-Latches
1× 48polige IC-Fassung (oder 2× 24polige)
4× 20polige IC-Fassung
1× 16polige IC-Fassung
6× 14polige IC-Fassung
1× 8polige IC-Fassung
R1, R2, R3, R4, R5, R11, R12 4.7 kΩ 1/8 W
R6, R7, R9, R10, R14, R15, R16, R17 1 kΩ 1/8 W
R8 100 kΩ
R13 47 Ω
C1, C7 10 µF 16 V
C2, C3, C4, C5, C6, C7 100 nF
S1 Minitaster
Q1 Quarz 8 MHz
D1 Zenerdiode ZY 5.1 oder 1N4002
JMP1 2× 8polige Stiftleiste, gerade
JMP4/5 2× 2polige Stiftleiste, gerade
St1 1 x 36polige Stiftleiste +1 x 18polige Stiftleiste, gewinkelt
3× Shuntstecker (für Jumpers).
1× Leiterplatte CPU68K

Kenndaten:
5-V-Versorgung, 430 mA Stromaufnahme.

Abb.7.3.1 zeigt das komplette Schaltbild der Baugruppe. Nachdem der Mikroprozessor 68008 nicht alle Signale unterstützt, die wir auf dem Bus brauchen, müssen verschiedene erst erzeugt, bzw. umgewandelt werden. *Abb. 7.3.2* zeigt einen Vergleich zwischen dem 68008 und dem Z80, der die Signale direkt liefern kann. Wie zu erkennen ist, gibt es eine ganze Reihe von Gemeinsamkeiten. Es existieren ein Adreß- und ein Datenbus. Der Adreßbus hat beim 68008 20 Leitungen, er kann damit einen größeren Speicher ansprechen als der Z80 mit seinen 16 Leitungen. Die Bus-Steuersignale unterscheiden sich schon mehr. Beim Z80 gibt es -MREQ, -IORQ, -RD und -WR, beim 68008 sind es die Signale -AS, R/-W und -DS. *Abb. 7.3.3* zeigt ein Umsetzungsschema. Das R/-W–Signal gibt an, ob gelesen oder geschrieben werden soll. Durch eine Verknüpfung mit -DS kann man die Signale -RD und -WR erzeugen. Die Funktion des Signals -IORQ ist mit der von -MREQ vergleichbar, nur daß es beim Zugriff auf IO-Adressen auftritt. Es wird immer dann erzeugt, wenn der 68008 auf Adressen im Bereich $F0000 bis $FFFFF zugreift. Damit sind die oberen 64 KByte nicht als Hauptspeicher verwendbar und für den IO-Bereich reserviert. Die Verknüpfung übernimmt IC 10, *Abb. 7.3.4* zeigt eine Wahrheitstafel dieses Schaltkreises.

So arbeitet die CPU:

Das IC 5, ein 74LS04, erzeugt den Takt. Die Quarzfrequenz beträgt 8 MHz. Man kann den Takt bei Bedarf halbieren, dazu dient die Brücke JMP4 und das IC 74LS74. Dies ist für ein paar Experimente interessant. Normalerweise könnte das IC 7, ein 74LS74, entfallen. Das IC hatte ursprünglich die Aufgabe, einen symmetrischen Takt zu erzeugen, wie er erst nach einer Halbierung entsteht. Nun ist glücklicherweise der Takt ab 8 MHz bereits ohne diese Schaltung ausreichend symmetrisch.

Die Reset- oder Startlogik wird durch das IC 1, einem NE 555, gebildet. Das IC eignet sich besonders zur Erzeugung einer präzisen Pulsdauer. Die -HALT und -RESET-Eingänge der CPU erhalten beide das RESET-Signal nach dem Spannungseinschalten, oder wenn man die RESET-Taste drückt. Da -RESET und -HALT auch Ausgänge der CPU sein können, werden sie über offene Kollektoren angesteuert (IC 2, 7405). Der -RESET-Ein-/Ausgang ist so verschaltet, daß er auch einen RESET am Bus auslösen kann. Dazu gibt es einen eigenen Mikroprozessorbefehl (RESET). Die beiden Eins-aus-vier-Dekoder im 74LS139 (IC 10) haben die Aufgabe, die Steuer-Signale für den Bus zu erzeugen. Das -IORQ-Signal wird immer dann erzeugt, wenn auf den Adreßbereich F0000 bis FFFFF zugegriffen wird; -MREQ dagegen erscheint im Bereich 00000 bis EFFFF. Die Peripherie-Baugruppen werten allerdings nur die Adressen A0 bis A7 aus, so daß anstelle der theoretisch möglichen 65536 IO-Adressen eigentlich nur 256 zur Verfügung stehen. Der gesamte Bereich kann durch eine eigene, zusätzliche Dekodierung zugänglich gemacht werden.

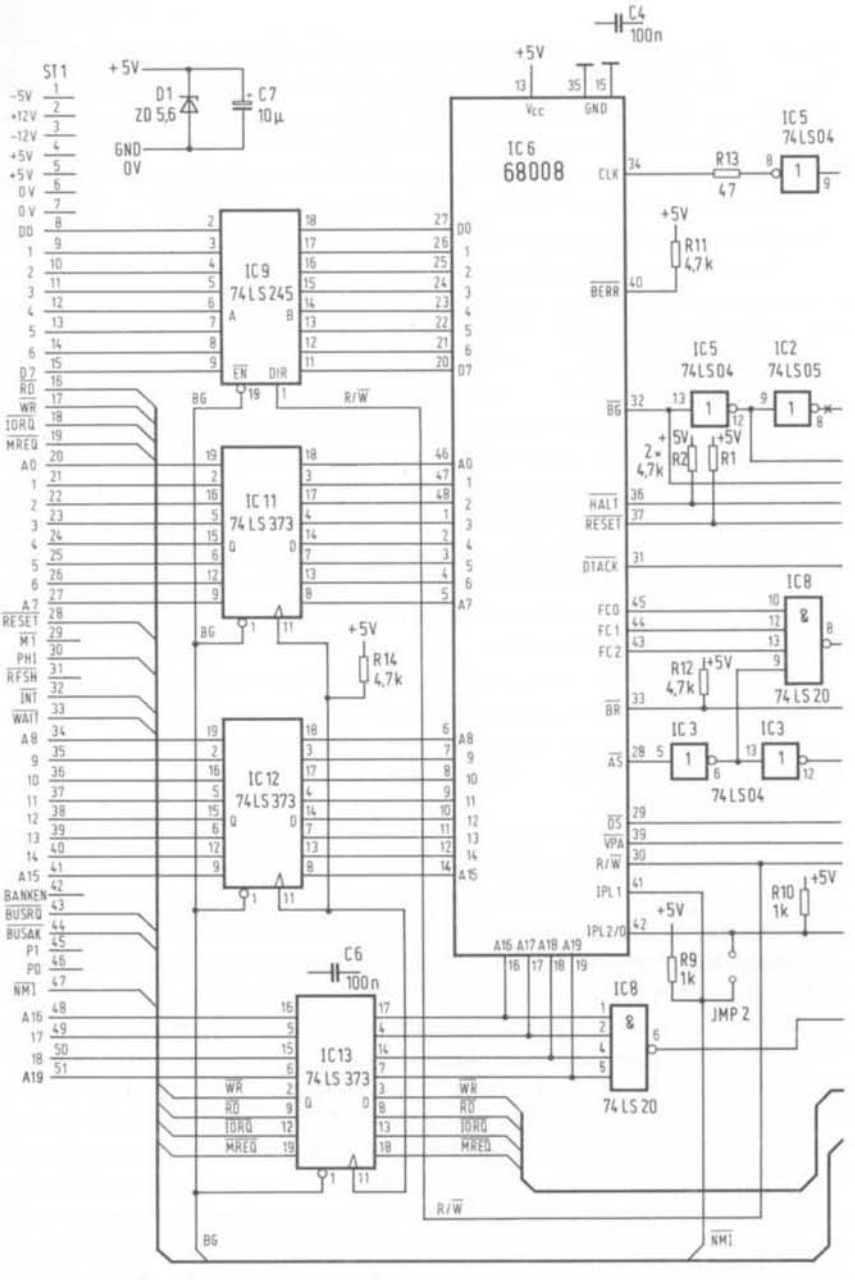
An die Dekoder ist zusätzlich das Signal -VPA geführt, es dient der Interrupt-Steuerung. Wir arbeiten nicht mit dem Vektor-Interrupt. Das Auftreten von FC0, FC1 und FC2 gemeinsam mit -AS bedeutet eine Interruptbestätigung; zu diesem Zeitpunkt muß durch -VPA der Auto-Vektor-Interrupt aktiviert werden.

Wartezyklus einstellbar

Mit dem IC 4 (74LS164) lassen sich Warte-Zyklen einstellen. Wenn man eine Brücke bei JMP1 einsetzt, wird jeder Zugriff auf den Speicher oder die Peripherie verlängert. JMP4 und JMP5 sind normalerweise immer eingesetzt. Der kürzeste Wartezyklus ist über die Brücke 1 zum Eingang A einstellbar, der längste wird über Brücke 8, Eingang H bewirkt. Ganz ohne Brücke erfolgen alle Zugriffe mit maximaler Geschwindigkeit.

Die CPU besitzt zur Steuerung der Wartezyklen einen Eingang mit dem Namen -DTACK, was soviel bedeutet „data ackknowledge" oder Daten-Akzeptierungs-Signal. Liegt dieser Eingang auf 0, was ohne Brücke über IC2 erfolgt, so wird jeder Zugriff mit maximaler Geschwindigkeit durchgeführt. Wenn man eine der Brücken einsetzt, ist das IC4 für die -DTACK-Erzeugung verantwortlich. Das Signal DS am Eingang CLEAR des Schieberegisters sorgt dafür, daß zunächst alle Ausgänge des Schieberegisters auf 0 liegen. Am -DTACK-Eingang liegt dann ein 1-Signal an. Wenn nun -DS auf 0 geht, wird der Eingang CLEAR am Schieberegister auf 1 gehen, und damit beginnt das Schieberegister, ein 1-Signal durchzuschieben. Je nach Brückenstellung gelangt es nun früher oder später an das IC5, Pin 5; über weitere Inverter gelangt es dann invertiert an -DTACK.

Nun gibt es langsame Peripherie oder Speicher, die für eine sichere Funktion eine höhere Anzahl von Wartezyklen benötigen. Um nicht alle Zugriffe langsamer machen zu müssen, gibt es ab Revision 4 der CPU-Baugruppe eine Schaltungslogik für das


ST1
-5V
+12V
-12V
+5V
+5V
0V
0V
0V
D0
1
2
3
4
5
6
D7
RD
WR
IORQ
MREQ
A0
1
2
3
4
5
6
A7
RESET
M1
PHI
RFSH
INT
WAIT
A8
9
10
11
12
13
14
A15
BANKEN
BUSRQ
BUSAK
P1
P0
NMI
A15
17
18
A19
+5V
D1
ZD 5,6
GND
0V
C7
10µ
C4
100n
C6
100n
IC 6
68008
Vcc
GND
CLK
BERR
BG
HALT
RESET
DTACK
FC0
FC1
FC2
BR
AS
DS
VPA
R/W
IPL1
IPL2/0
A16 A17 A18 A19
IC 9
74LS245
A
B
EN
DIR
IC 11
74 LS 373
IC 12
74 LS 373
IC 13
74 LS 373
IC 5
74LS04
IC 2
74LS05
IC 3
74LS04
IC 8
74 LS 20
R13
47
R11
4,7k
R14
4,7k
2 x
4,7k
R2
R1
R12
4,7k
R10
1k
R9
1k
JMP 2
C6
100n

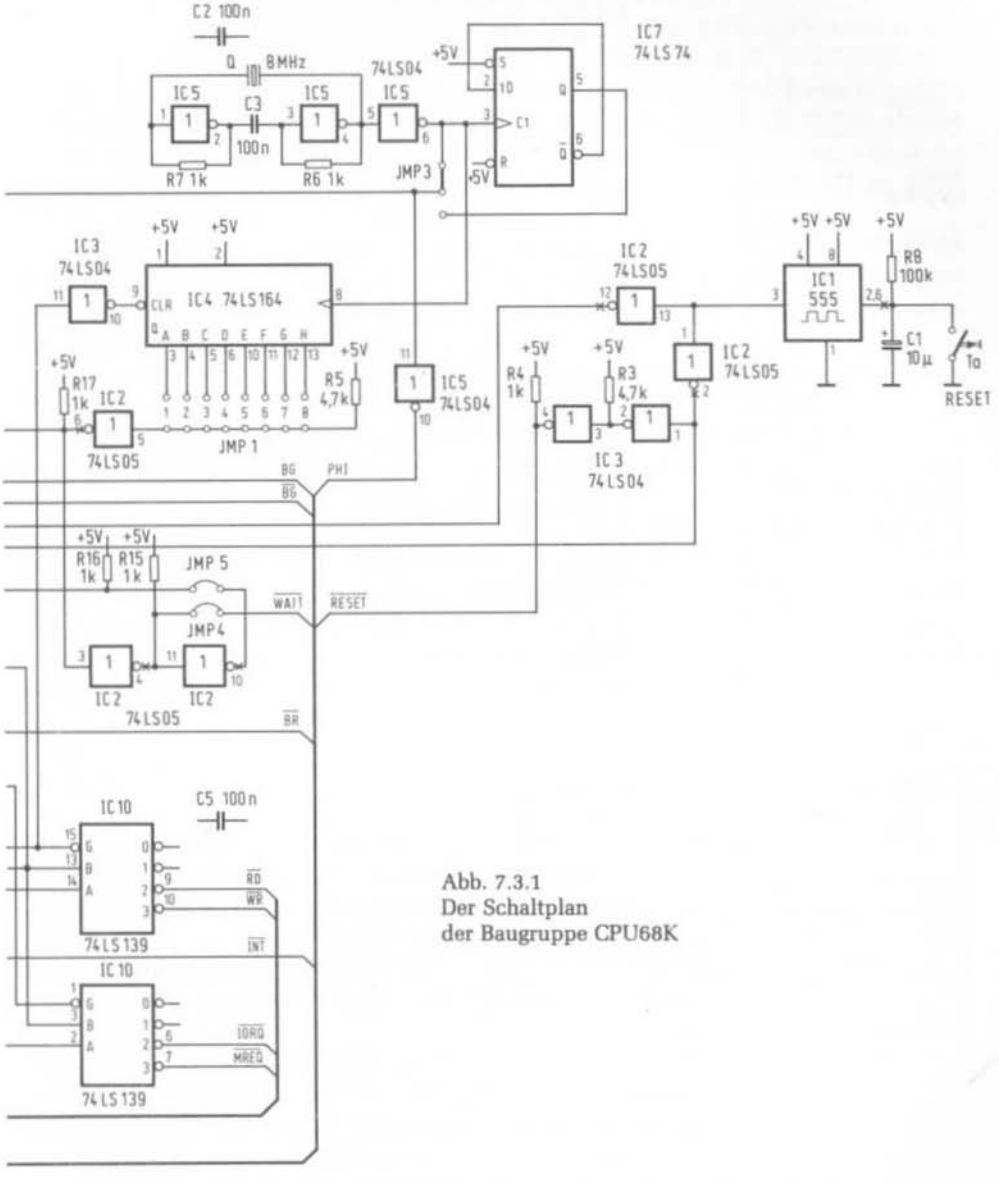


Abb. 7.3.1
Der Schaltplan
der Baugruppe CPU68K

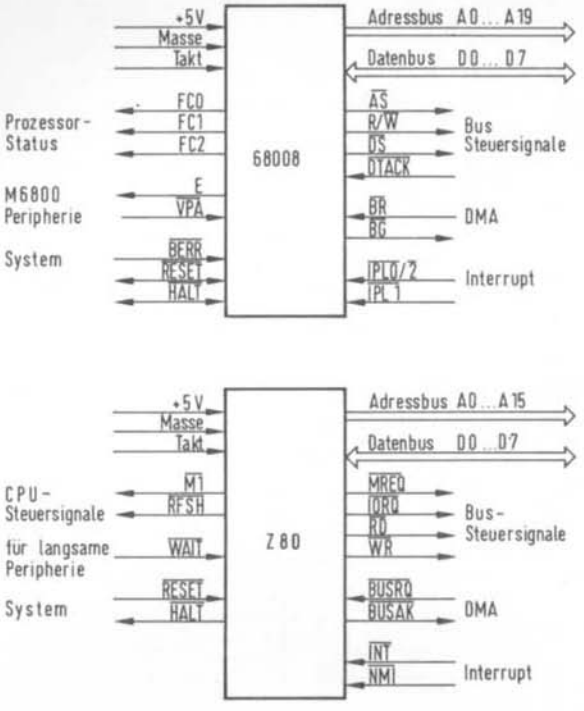


Abb. 7.3.2
Signalvergleiche
68008 – Z80

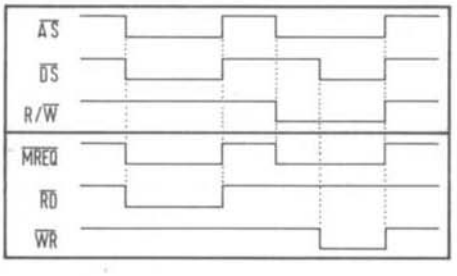


Abb. 7.3.3
Die Herleitung
der Bus-Signale

$\overline{G}$	B	A	0	1	2	3
0	0	0	0	1	1	1
0	0	1	1	0	1	1
0	1	0	1	1	0	1
0	1	1	1	1	1	0
1	x	x	1	1	1	1

Abb. 7.3.4
Die Wahrheitstafel
des Bausteins 74LS139

-WAIT-Signal am BUS. Wenn dieses Signal auf 0 liegt, soll die CPU mindestens so lange warten, bis es wieder auf 1 geht. Das Signal gelangt dazu über JMP4, der normalerweise gesteckt sein muß, an das IC2, Pin 11. Wenn das Signal 0 ist, so wird -DTACK auf 1 gehalten. Damit ist auch die Wartezyklen-Schaltung für die Zeit unwirksam. Erst wenn WAIT wieder auf 1 zurückgeht, hängt es von der Brückenstellung an JMP1 ab, ob jetzt -DTACK auf 0 geht oder noch nicht. IC2 ist ein Baustein mit offenen Kollektoren in der Ausgangsschaltung; dies ist unbedingt erforderlich, da der Ausgang -WAIT an einen weiteren Ausgang (Pin 4, IC2) geschaltet wird.

Nun gibt es noch eine Brücke, die noch nicht besprochen wurde: JMP2. Wenn sie eingesetzt ist, so werden -INT und -NMI zusammengeschaltet. Damit wird beim 68008 ein nichtmaskierbarer Interrupt höchster Priorität erzeugt. Normalerweise sind beide Interrupts maskierbar und müssen also per Programm erst freigegeben werden, um wirksam zu sein. Der nichtmaskierbare Interrupt ist außerdem auf Flanken empfindlich, während die beiden anderen nur auf statische Pegel reagieren.

Eine Neuerung in der Revision 4 der Baugruppe ist auch die DMA-Fähigkeit. Wenn das Signal -BUSRQ auf 0 V gelegt wird, gibt der 68008 den Bus frei. Er gibt die Bestätigung über die Leitung -BUSAK, die dann auf 0 V geht, aus. Alle Signale mit Ausnahme des Betriebstaktes nehmen dann Tri-State an. Wenn -BUSRQ wieder auf 1 zurückgeht, beginnt die CPU wieder zu arbeiten. Im Prinzip kann man damit nicht nur DMA-Bausteine, sondern auch andere CPU-Baugruppen anschließen; außer den besprochenen Dingen ist natürlich zu beachten, daß immer nur ein Takt auf den Bus gelegt wird. Die Erzeugung von -BUSRQ muß durch eine Zusatzschaltung erfolgen; es darf immer nur eine von beiden CPUs ein 0-Signal erhalten.

Brücken:

Hier nochmals eine kurze Zusammenfassung der Brückeneinstellungen:

JMP1 Einstellen der Warte-Zyklen. Bei langsamen EPROMs muß mindestens Position 1 eingestellt sein, beim Betrieb mit der FLO2-Baugruppe muß Position 2 eingestellt werden.

JMP2 Normalerweise offen. Wenn der nichtmaskierbare Interrupt verwendet werden soll, wird eine Brücke eingesetzt. Die Eingänge -INT und -NMI haben dann die gleiche Bedeutung.

JMP3 gibt es bei Revision 4 nicht mehr.

JMP4 Durchschaltung der -WAIT-Leitung, normalerweise eingesetzt.

JMP5 Durchschaltung des -DTACK-Signals, normalerweise eingesetzt. JMP5 kann so gesteckt werden, daß das Signal -DTACK direkt auf den Bus an -WAIT führt. Dies ist nur für Experimente zu verwenden, um Kompatibilität zu garantieren.

Abb. 7.3.5 zeigt die Lötseite, *Abb. 7.3.6* die Bestückungsseite der Baugruppe. In *Abb. 7.3.7* ist der Bestückungsplan zu sehen.

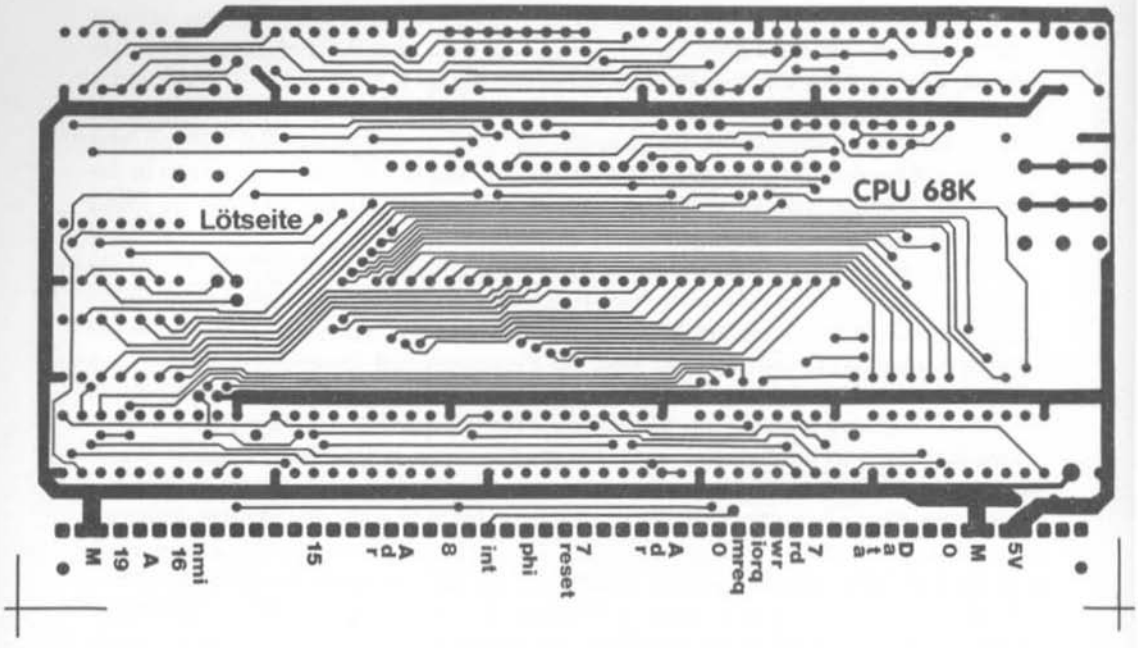


Abb. 7.3.5 Die Lötseite der Leiterplatte CPU68K

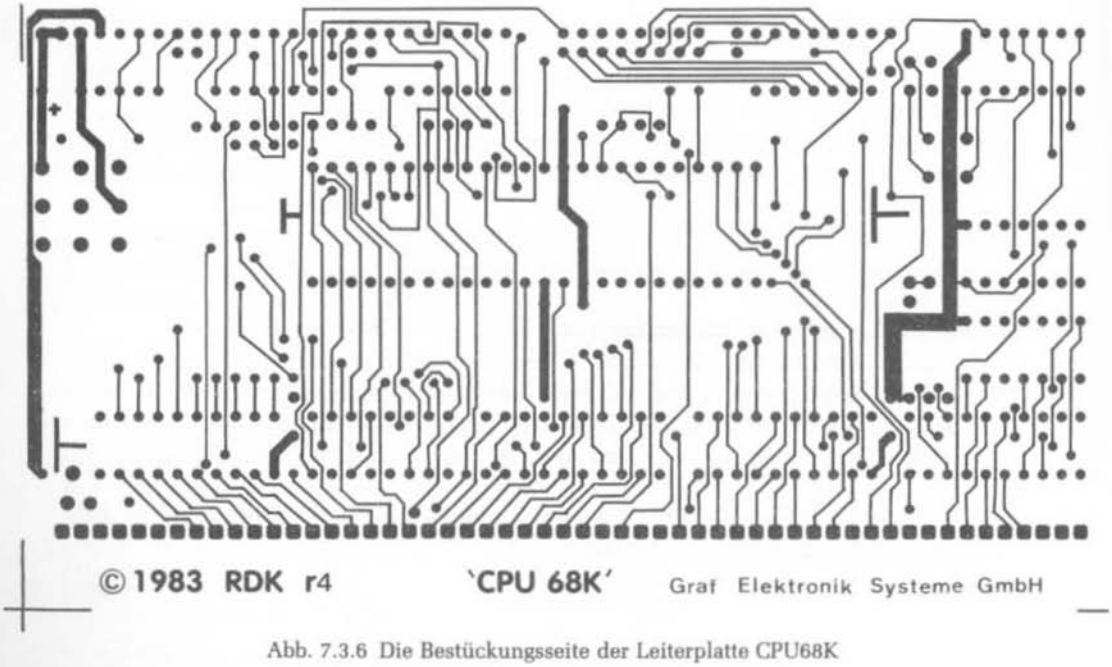


Abb. 7.3.6 Die Bestückungsseite der Leiterplatte CPU68K

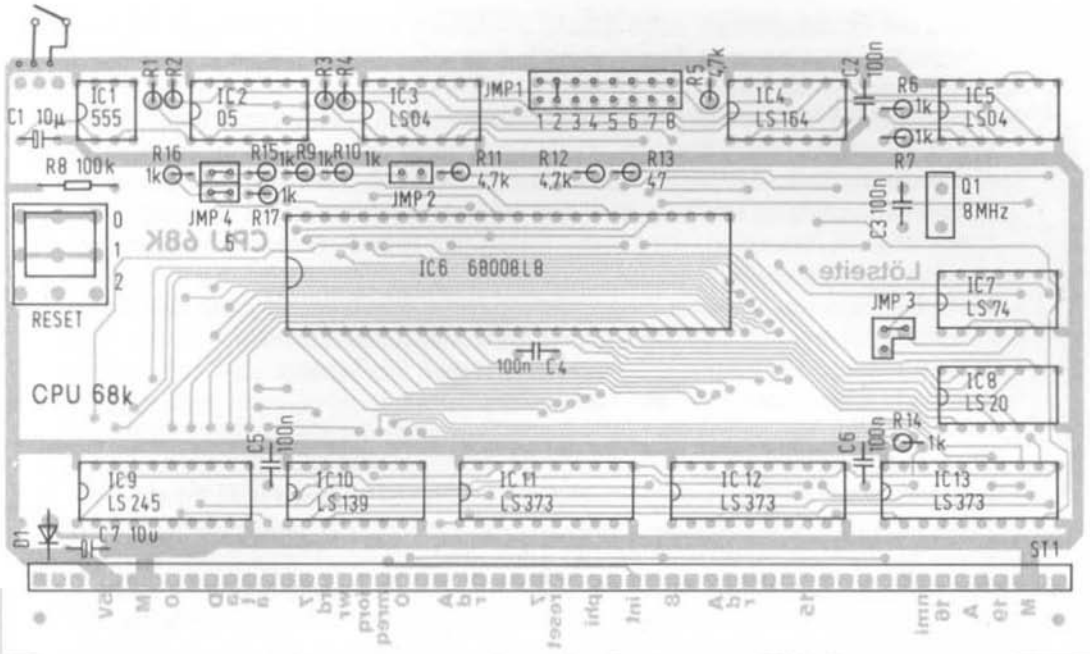


Abb. 7.3.7 Der Bestückungsplan der Baugruppe CPU68K

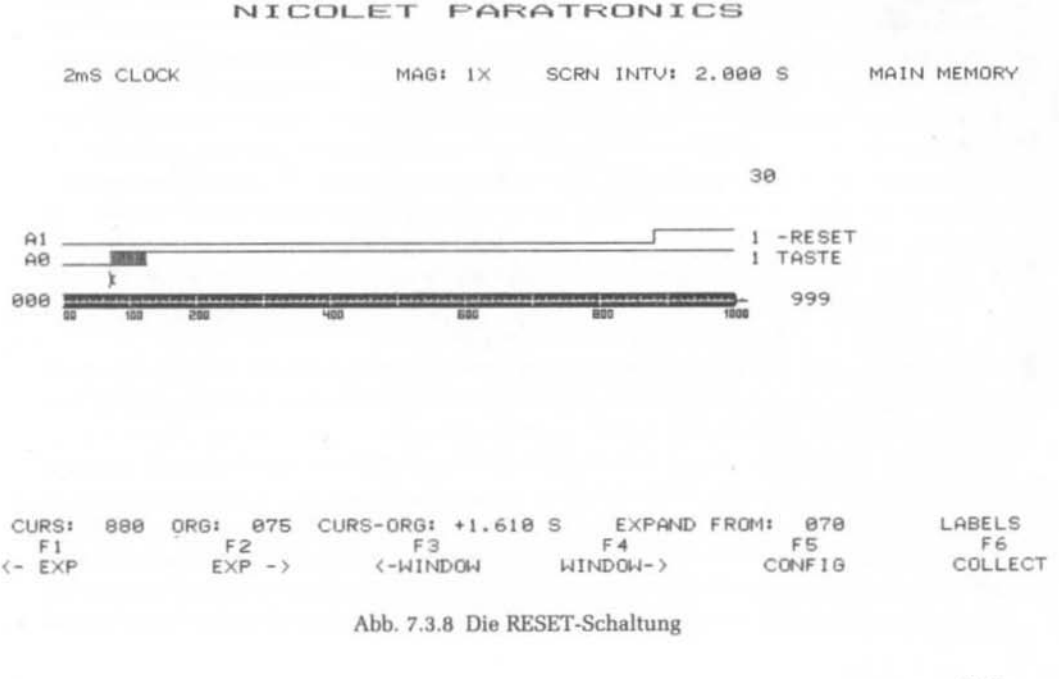


Abb. 7.3.8 Die RESET-Schaltung

Aufbau der Baugruppe:

1. Einlöten aller Sockel und aller passiven Bauteile.
2. Test: Anschluß an die Versorgungsspannung. Jetzt kann man Kurzschlüsse noch am besten feststellen. Prüfen Sie auch, ob die Versorgungsspannung richtig gepolt ist. Ist dies nicht der Fall, so erzeugt die eingebaute Zenerdiode einen Kurzschluß, um die restlichen Bauteile zu schützen. Ebenfalls schützt sie kurze Zeit vor Überspannung.

   Achtung: die Zenerdiode wird auch im normalen Betrieb warm, da die Zenerspannung recht knapp gewählt sein muß, um überhaupt eine Wirkung zu zeigen. Wen das stört, der kann sie entweder ganz entfernen (kein Schutz mehr), oder durch eine normale Leistungsdiode ersetzten (kein Schutz vor Überspannung).

   Messen Sie an Pin 13 des CPU-Sockels, dort muß +5 V liegen.
3. Spannung ausschalten.
4. Einsetzen aller ICs, außer dem CPU-Baustein. Prüfen Sie nochmals sorgfältig die Orientierung der einzelnen ICs; verpolte ICs können defekt werden.
5. Test: schalten Sie die Spannung ein, und messen Sie an Pin 34 der CPU; dort muß ein Takt von 8 MHz ankommen.

Messen Sie an Pin 36 und Pin 37 der CPU, dort erscheint das -RESET-Signal, während man die RESET-Taste drückt.

Abb. 7.3.8 zeigt ein Timing-Diagramm dazu. Es wurde mit einem Logik-Analysator erzeugt, um die Darstellung klarer zu gestalten. Sie finden in der oberen Zeile die Angaben über Abfragetakt (hier 2 ms), über die Vergrößerung des Bildes, hier 1×, und über die Zeitdauer, die auf dem Bildschirm dargestellt ist, hier 2 s. In der Mitte ist das Timing abgebildet, unten sind weitere Zusatzangaben zu sehen. Das Signal „TASTE" ist direkt am Taster abgenommen. Diese Taste wurde gedrückt und dann losgelassen. Das Loslassen sieht man ganz links im Bild, das Signal geht auf 1. Die etwas unklare Pulsflanke des Signals kommt daher, daß es langsam ansteigt und kein TTL-Signal ist. Oben ist das -RESET (und -HALT)-Signal dargestellt. Es erscheint ca. 1.6 s nach dem Loslassen der Taste. Die Zeitdauer ist völlig unkritisch, sie kann bei Ihnen etwas anders ausfallen. Wenn Sie später einmal die dynamische Speicherbaugruppe einsetzen wollen, können Sie anstelle des 100 kΩ Widerstandes auch einen kleineren Wert verwenden, um die Zeitdauer zu verkürzen. Während des Reset erfolgt kein Refresh, aus diesem Grund darf dieser Zeitwert nicht zu groß eingestellt werden.

Mit dem Prüfstift kann man die Pulse sehr schön sehen. Die Leuchtdioden W1 und W2 wechseln außerdem bei jedem Reset.

6. Abschalten und dann die CPU 68008 einsetzen.
7. Test: wenn Sie schon alle anderen Baugruppen aufgebaut haben, könnten Sie einen Gesamttest wagen. Dabei empfiehlt sich folgende Kombination: CPU68K + GDP64 + ROA64 (mit Grundprogramm und einem RAM).

Auf dem Bildschirm muß sich auch ohne KEY-Baugruppe schon das Grundprogramm melden. Dann ist die CPU in Ordnung.

Sollten Sie nicht zu den Glücklichen gehören, so finden Sie hier ein paar nützliche Zeitdiagramme. *Abb. 7.3.9* zeigt den Vorgang nach dem Einschalten, wenn nur die CPU auf dem Bus sitzt. 1,8 µs nachdem das -RESET-Signal auf 1 ging, werden 8 Byte aus den ersten Speicherzellen des Speichers gelesen. Die CPU lädt hier den Stackpointer und

dann die Startadresse (PC). Da in unserem Testfall keine ROA-Baugruppe mit EPROMs eingesetzt ist, sind die Aktionen nach diesen Zugriffen undefiniert. Während der Aufnahme der Abbildung erfolgte ein Zugriff auf die IO, jedoch nur mit einem -IORQ-Signal; danach blieb die CPU stehen. Nur die ersten 8 Zugriffe müssen bei jedem System gleich aussehen.

Für die weiteren Messungen ist es erforderlich, das System mit einer ROA64-Baugruppe und dem Grundprogramm zu komplettieren. Wie geht man aber meßtechnisch vor?

Man klemmt den Triggereingang an das -RESET-Signal, so daß der Trigger ausgelöst wird, wenn eine positive Flanke erscheint. Mit dem Y-Eingang kann man nun nacheinander die Signale beobachten. Da der Strahl nur einmal durchläuft, ist das Bild sehr dunkel. Dies kann man ändern, wenn man den NE555 aus der Fassung nimmt und dort an Pin 3 einen externen, ca. 100-kHz-Takt legt. Dann erfolgt der Reset zyklisch, und man hat ein periodisches Signal, das man gut beobachten kann.

Kontrollieren Sie alle Leitungen auf Kurzschlüsse; man erkennt Sie daran, daß die Pegel um 1 V liegen. Achtung: die Signale in diesem Spannungsbereich nicht mit dem Tri-State verwechseln! Die Signale sehen auf dem Skop nicht so schön rechteckig aus, das stört jedoch nicht weiter.

Abb. 7.3.10 zeigt einen groben Ablauf nach dem Reset im Zeitraum von 100 µs, bei aktivem Grundprogramm. Nach einer gewissen Zeit erfolgt ein Schreib-Zugriff. Sie können das mit dem Skop bei entsprechender Zeitbasis-Einstellung kontrollieren. Dabei kommt es auf die genaue Pulsform nicht an.

Abb. 7.3.11 zeigt nochmals die Zeitrelationen zwischen -DS, -AS und den Bussignalen -MREQ, -IORQ, -WR und -RD. Das Diagramm wurde während der Arbeit mit dem Grundprogramm aufgenommen. Es durchlief eine Schleife, welche sich mit einem Skop ebenfalls beobachten läßt.

Abb. 7.3.12 entstand bei einer Einstellung von zwei Wartezyklen (Position 2 an JMP1) und soll das Zeitverhalten des -DTACK-Signals verdeutlichen. Stellt man eine noch höhere Position ein, verschiebt sich das -DTACK-Signal, bezogen auf die abfallende -DS-Flanke, noch weiter nach rechts.

Damit sind der Test und die Fehlersuche abgeschlossen.

7.4 Die CPU68K/16

Dieser Abschnitt ist für Fortgeschrittene gedacht. Die CPU 68000 arbeitet mit einem 16-Bit-Datenbus. Damit ergibt sich eine Geschwindigkeitssteigerung gegenüber dem 68008, da nun in der gleichen Zeit 2 Byte über den Datenbus transportiert werden können. Ferner kann man den Takt erhöhen; es gibt den 68000 bereits für eine Frequenz von 12MHz, bald wird es ihn für 16MHz geben.

Hier nochmals die Warnung: der NDR-KLEIN-Computer sollte nur von erfahrenen Nachbauern mit der 68000-CPU ausgerüstet werden, da sich viele Tests komplizierter gestalten und man auch den doppelten Speicher benötigt (alles doppelt so teuer wie beim 68008). Außer den in diesem Kapitel aufgezeigten Hardwareabweichungen muß

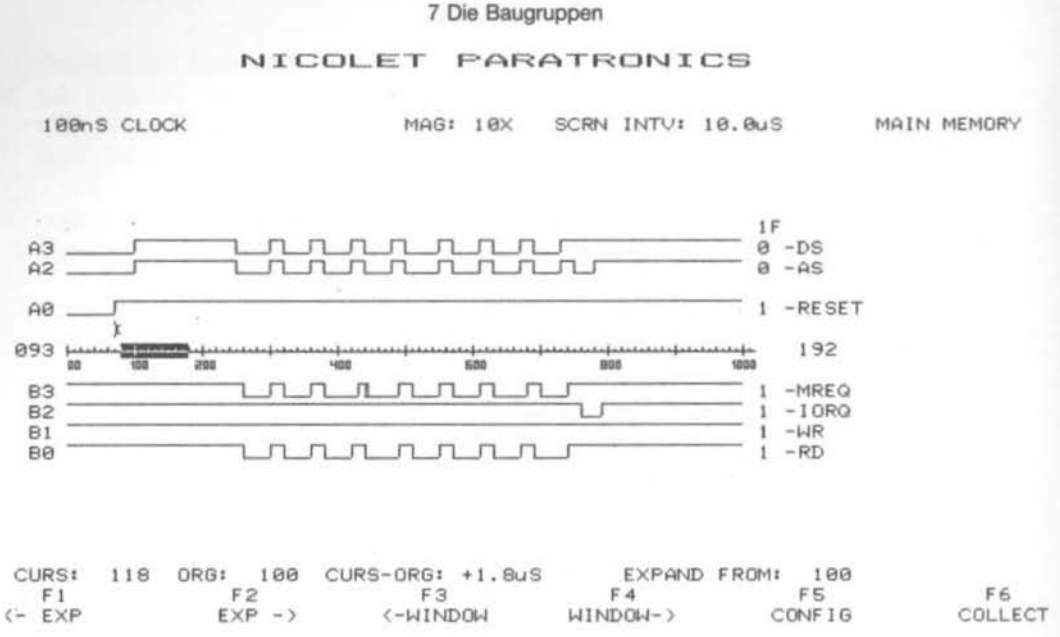


Abb. 7.3.9 Die Erzeugung der Bus-Signale

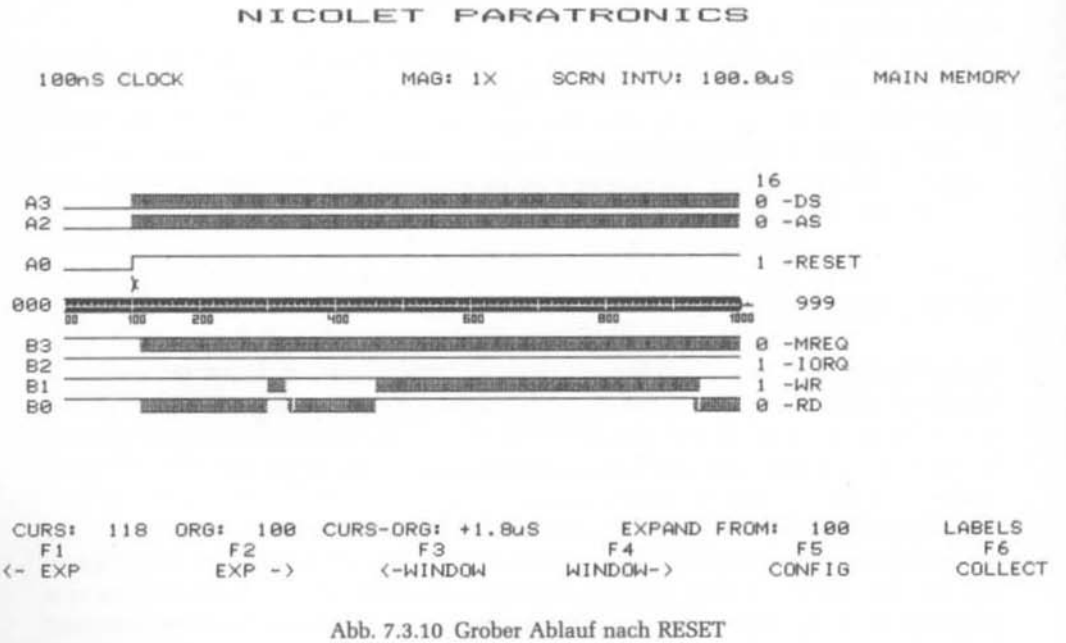


Abb. 7.3.10 Grober Ablauf nach RESET

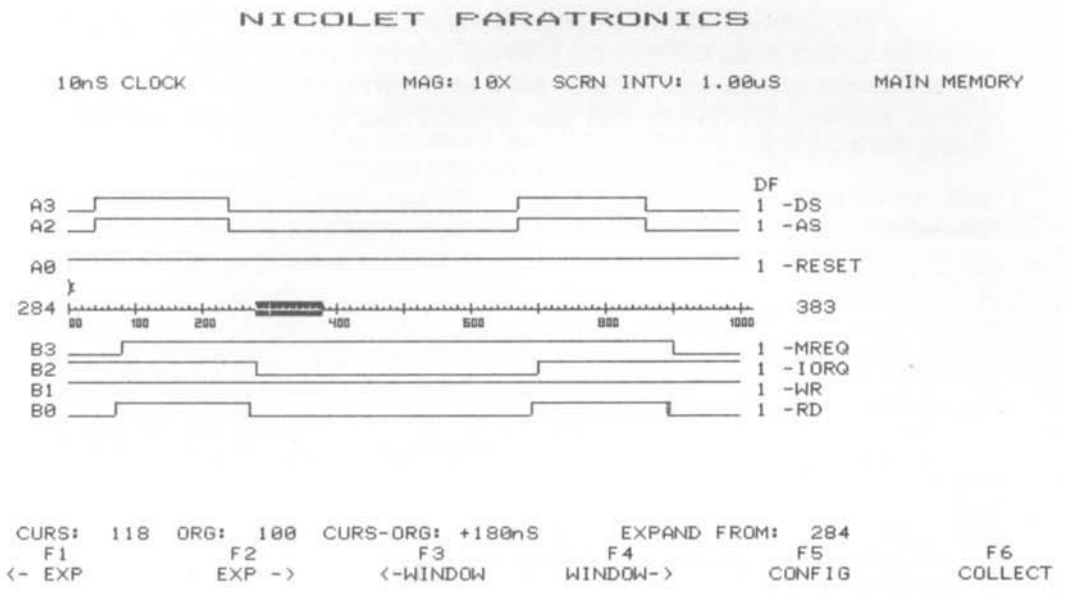


Abb. 7.3.11 In der Grundprogramm-Schleife

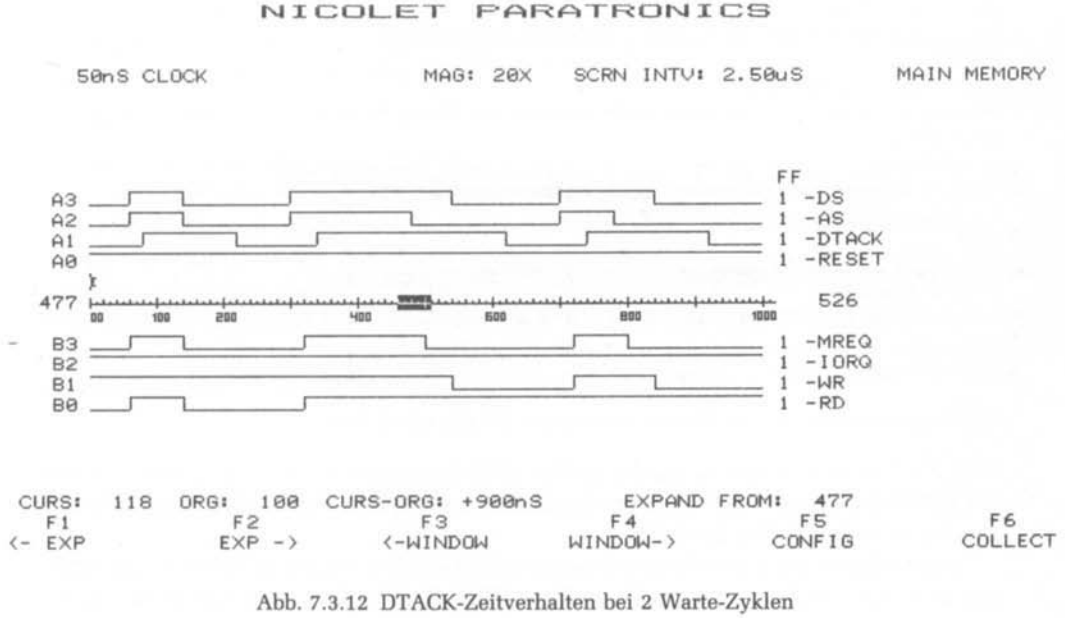


Abb. 7.3.12 DTACK-Zeitverhalten bei 2 Warte-Zyklen

man auch das entsprechende Grundprogramm einsetzen, das für den 68000 übersetzt wurde (im Listing, wenn CPU=2). Die EPROMs sind im Handel erhältlich.

Alle Versuche müssen dann mit einer mit 2 multiplizierten IO-Adresse durchgeführt werden, also statt $FFFFFF30 muß man $FFFFFF30*2 schreiben (das Ausrechnen besorgt der Assembler).

Materialliste:

IC1,IC2 74LS245
IC3,IC4,IC5 74LS373
IC6,IC9 74LS139
IC7 MC 68000 L8 oder L12
IC8 74LS00
IC10 74LS20
IC11 7404
IC12 74LS164
IC13 NE 555
IC14 7405
IC15 74LS04
1× 64polige IC-Fassung (oder anreihbare Buchsen verwenden).
5× 20polige IC-Fassung
2× 16polige IC-Fassung
6× 14polige IC-Fassung
1× 8polige IC-Fassung
Q1 Quarz 8 MHz (oder 12 MHz bei L12-Version der CPU)
AR1 Widerstandsnetzwerk 8× 3.3 kΩ oder 4.7 kΩ (Wert unkritisch).
R1, R2 1 kΩ
R3 47 Ω
R4 100 kΩ
R5, R6, R7 4.7 kΩ
C1, C3, C4, C5, C6, C7, C8, C10, C11, C12, C13, C14, C15
100 nF
C2, C9 10 µF Tantal, 16 V
Achtung, C3 wird auf die Rückseite direkt zwischen Pin 53 und Pin 49 der CPU gelötet! Dafür gibt es keine Löcher.
1× Minitaster.
1× doppelreihige 54polige gewinkelte Steckerleiste (oder zusammengesetzt aus 26 + 1x18polig).
1× doppelreihige 8polige gerade Steckerleiste für
JMP1
1× 2polige Stiftleiste für J (-INT und -NMI Verbindung).
2× Shuntstecker
1× Leiterplatte CPU 68K/16

Kenndaten:

Spannungsversorgung +5 V, Stromaufnahme: ca. 530 mA bei 12 MHz

Abb. 7.4.1 zeigt den Schaltplan der großen CPU-Baugruppe. Im Prinzip arbeitet sie wie die 68008-Baugruppe, nur ist eben der Datenbus doppelt so breit. Aus diesem Grund werden zwei Bustreiber benötigt.

Ferner müssen auch die Steuerleitungen geteilt werden. So gibt es hier z.B. ein -WRL und ein -WRU, außerdem ein -RDL und ein -RDU. L steht für Lower und U für Upper.

Diese Teilung ist nötig, damit die CPU bei Byte-Zugriffen auf einzelne Hälften des Busses zugreifen kann. Die Teilung wird durch die Signale -LDS und -UDS gesteuert, die die alte Funktion -DS beinhalten.

Die Baugruppe ist in der hier gezeichneten Form nicht DMA-fähig, da kein Platz mehr für einen zusätzlichen Treiber vorhanden war. Die Einstellung der Wartezyklen sowie das restliche Timing funktionieren genauso wie bei der kleinen CPU.

Dadurch, daß die Adresse A1 der CPU an den BUS A0 geführt ist (und bei A2...A20 entsprechend), kann man 2 MByte Speicher adressieren. Die IO-Baugruppen liegen im Bereich ab Adresse $F00000 bis $FFFFFF. Damit ist der Bereich von 2 MByte voll für RAM-Speicher verfügbar. Sicherheitshalber sollte man die IOs in den Bereich $FFFFFF00·2 bis $FFFFFFFF·2 legen. Dazwischen liegen die jeweils um eins erhöhten Adressen für die ungeraden IO-Adressen, sie sprechen die auf der rechten Bushälfte liegenden Baugruppen an (siehe BUS-Baugruppen-Beschreibung).

Abb. 7.4.2 zeigt die Löt-, *Abb. 7.4.3* die Bestückungsseite der Baugruppe. In der *Abb. 7.4.4* ist der Bestückungsplan abgedruckt.

Aufbau:

Man geht dabei genauso vor, wie bei der CPU68K mit dem 68008. Das Timing ändert sich nur geringfügig, jedoch folgen nach erfolgtem Reset nur 4 Lesezyklen, denn pro Zyklus wird nun ein ganzes Wort (zwei Bytes) gelesen.

Der Kondensator C3, mit 100 nF (im Schaltplan nicht eingezeichnet), darf nicht vergessen werden. Er muß auf der Lötseite direkt zwischen die Pins 49 und 53 der CPU 68000 gelötet werden. Wird er weggelassen, so kann es beim Betrieb mit 12 MHz zu Störungen kommen.

Zum Betrieb mit dem Grundprogramm sehen Sie im Kapitel 1.3 nach. Achten Sie auf die richtige Lage und Anordnung der EPROMs!

7.5 Die Speicherbaugruppe ROA64

Genauso wichtig wie die CPU ist natürlich auch der Speicher. Die ROA64-Baugruppe kann sowohl EPROMs, also Bausteine mit festprogrammierten Daten, wie auch statische Speicher aufnehmen. Dabei können die einzelnen Bausteine eine Kapazität von 8 K x 8 Byte besitzen, insgesamt kann die Baugruppe 64 KByte aufnehmen.

Auf diese Baugruppe wird auch das Grundprogramm in EPROMs gesteckt, sie ist also zum Betrieb des NDR-KLEIN-Computers unbedingt nötigt.

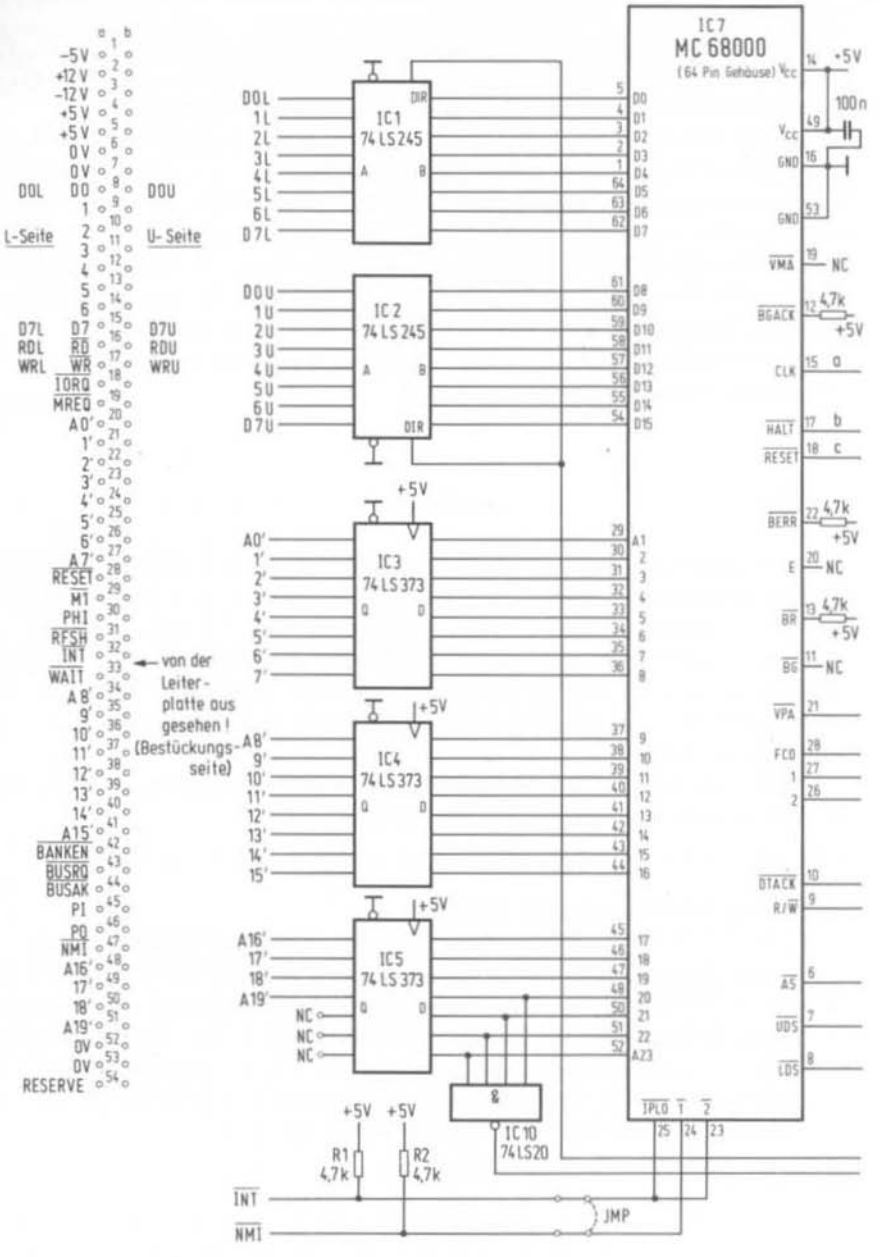
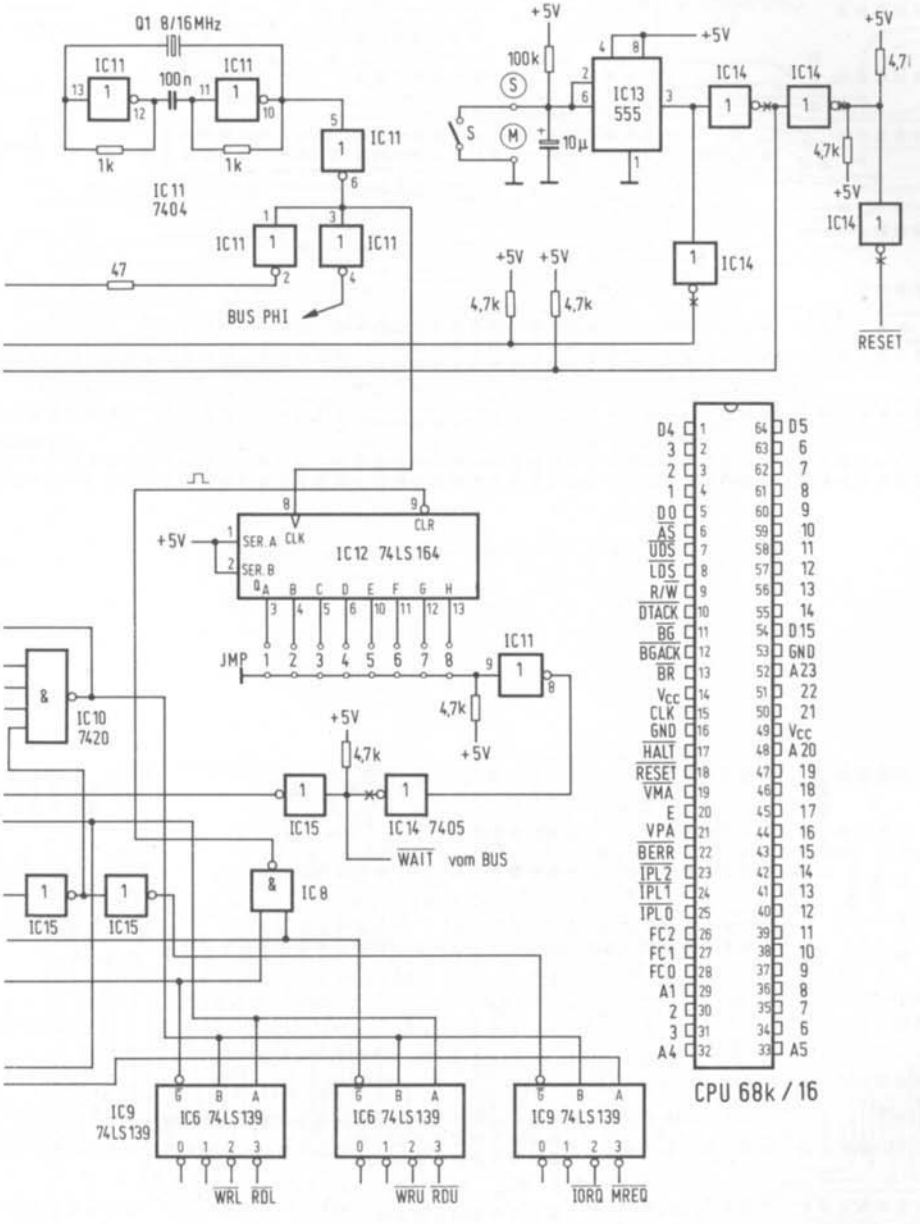


Abb. 7.4.1 Schaltplan der CPU68K/16


Q1 8/16MHz
IC11
100n
1k
IC11 7404
BUS PHI
47
+5V
100k
10µ
IC13 555
IC14
4,7k
RESET
IC12 74LS164
SER.A
SER.B
CLK
CLR
JMP
IC10 7420
IC15
IC14 7405
WAIT vom BUS
IC8
IC9 74LS139
IC6 74LS139
WRL RDL
WRU RDU
IORQ MREQ
D4 3 2 1 D0 AS UDS LDS R/W DTACK BG BGACK BR Vcc CLK GND HALT RESET VMA E VPA BERR IPL2 IPL1 IPL0 FC2 FC1 FC0 A1 2 3 A4
D5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 Vcc A20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 A5
CPU 68k / 16

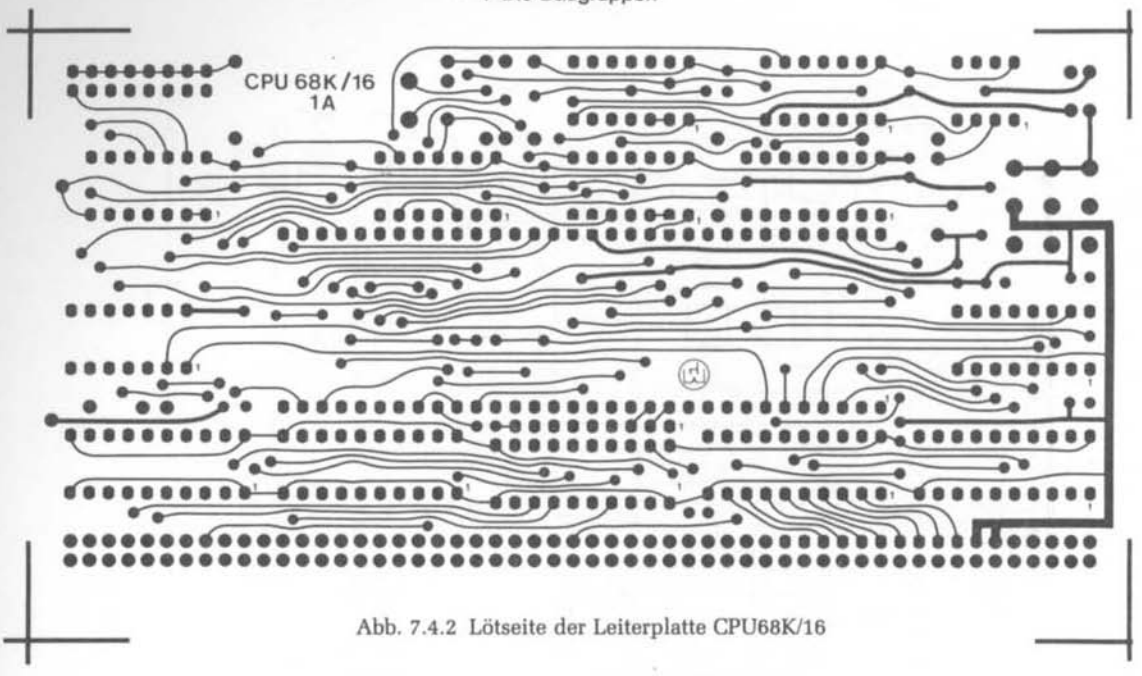


Abb. 7.4.2 Lötseite der Leiterplatte CPU68K/16

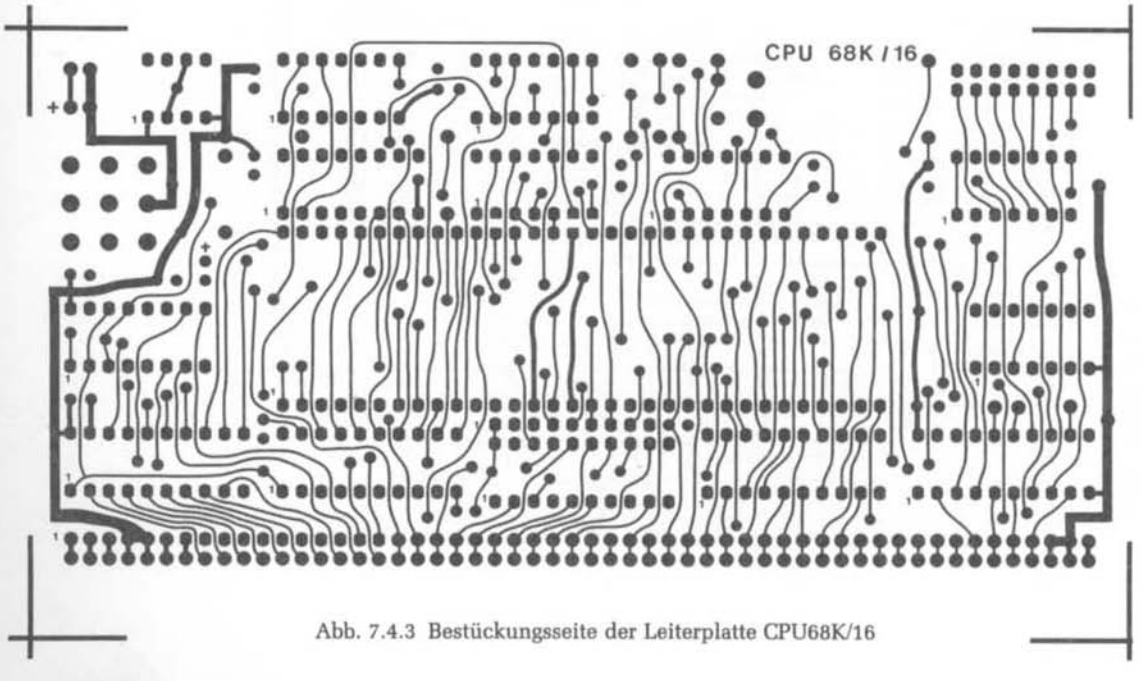


Abb. 7.4.3 Bestückungsseite der Leiterplatte CPU68K/16

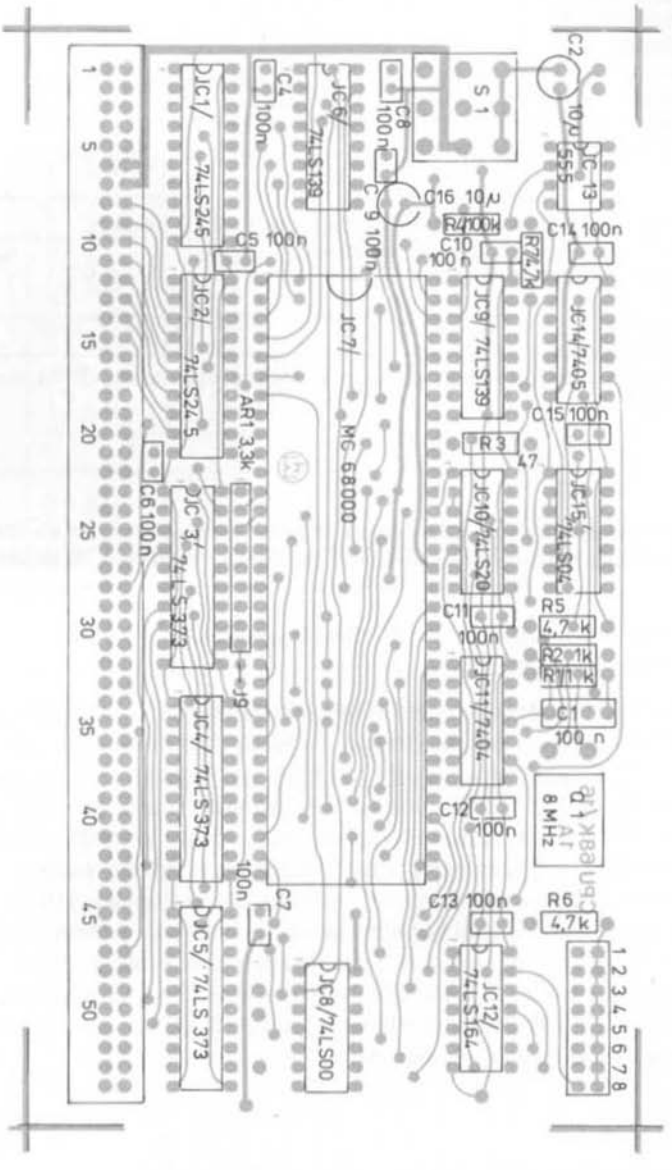


Abb. 7.4.4 Bestückungsplan der CPU68K/16

Materialliste:

IC9 74LS245, bidirektionaler Datenbustreiber
IC10 74LS138, 1 aus 8 Dekoder
IC11 74LS00, Nand-Glieder
IC12 74LS85, Vergleicher
8× 28polige Fassung
1× 20polige Fassung
2× 16polige Fassung
1× 14polige Fassung
D1 Zenerdiode ZY 5.1 oder Diode 1N4002
C1, C2 100 nF
C3 10 µF, 16 V
R1, R2, R3, R4, R5 1 kΩ, 1/8 W, Wert sehr unkritisch (auch 4.7 kΩ möglich).
St1 1× 36polige und 1× 18polige Stiftleiste, gewinkelt
JMP1 doppelreihige Stiftleiste, gerade
4× Shuntstecker
1× Leiterplatte ROA64
Optional je nach Anwendung :
EPROMS vom Typ 2764 mit 250 ns oder schneller. RAMs vom Typ HM 6264 P-15, oder TC 5565 P-15, oder äquivalente.

Kenndaten:

Spannung: +5 V; Stromaufnahme leere Baugruppe: 120 mA; Stromaufnahme mit 4 × EPROM 2764 + 1 × RAM: 200 mA (weitere RAMs haben kaum Einfluß auf die Stromaufnahme, da es CMOS-Bausteine sind).

Abb. 7.5.1 zeigt das Schaltbild der ROA64-Baugruppe.
Arbeitsweise der Schaltung:

Das IC 9, ein 74LS245, trennt die Datenleitungen D0 bis D7 des Busses von denen der integrierten Schaltkreise auf der Baugruppe. Dadurch wird der Bus nur gering belastet.

Die Auswahl der Baugruppe geschieht mit Hilfe des Bausteins IC 12, einem 74LS85. Dieser Baustein vergleicht die Signale an den Adreßleitungen A16 bis A19 mit der Einstellung an JMP2. Dort können Brücken nach Masse gesteckt werden, um die Adresse der Baugruppe festzulegen. Eine eingesteckte Brücke entspricht einem 0-Signal. Wenn alle Brücken gesteckt sind, müssen die Leitungen A16 bis A19 ein 0-Signal führen, damit der Vergleicher aktiv wird. Er liefert an seinem Ausgang (Pin 6) nur dann ein 1-Signal, wenn auch noch der Eingang (Pin 3) ein 1-Signal führt. Dieser Eingang liegt an der Busleitung BANKEN. Normalerweise liegt sie über den Widerstand R5 auf +5 V. Sie wird z.B. durch die Baugruppe BANK/BOOT auf 0 gelegt, wenn die ROA-Baugruppe ausgeschaltet werden soll.

Beispiele für Adreßeinstellungen:

Bereich 00000 bis 0FFFF A19, A18, A17, A16 gesteckt.
Bereich 10000 bis 1FFFF A19, A18, A17 gesteckt.

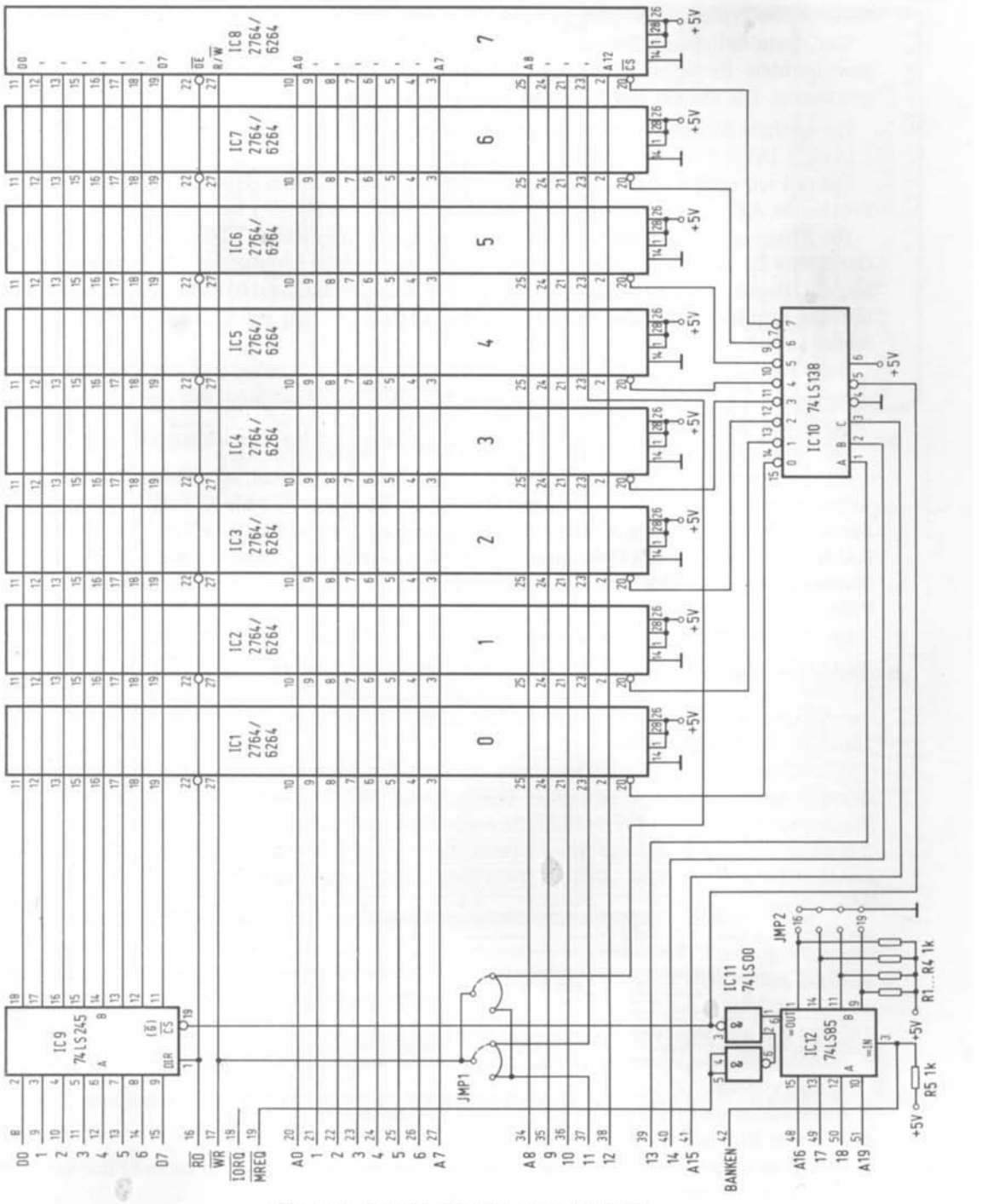


Abb. 7.5.1 Schaltplan der Baugruppe ROA64

Bereich E0000 bis EFFFF A16 gesteckt.

Um einen beliebigen Bereich zu ermitteln, schreibt man die oberen vier Bits des gewünschten Bereichs in dualer Schreibweise, also z.B. D0000 bis DFFFF ist gewünscht. Die oberen vier Bits sind sedezimal D oder dual 1101.

Die höchste Stelle entspricht in diesem Beispiel A19, also
1 (A19) 1 (A18) 0 (A17) 1 (A16)

Überall wo eine 0 steht, muß eine Brücke eingesetzt werden. Hier ist das also die Brücke bei A17.

Der Ausgang des Vergleichers führt an den Eingang eines Nand-Gliedes, IC 11, Pin 1. Der zweite Eingang dieses Nand-Gliedes wird vom ebenfalls invertierten -MREQ-Signal bedient. Damit liegt der Ausgang von IC 11, Pin 3 immer dann auf 0, wenn die Adresse mit den Brücken JMP2 übereinstimmen, das BANKEN-Signal auf 1 und das -MREQ-Signal auf 0 liegt.

Dann wird der Eingang des Bustreibers IC 9, Pin 19 an 0 V gelegt und der TRI-State aufgehoben. Die Datenrichtung wird durch das -RD-Signal bestimmt, das dazu an den Eingang DIR (direction = Richtung), Pin 1, IC 9 führt.

Ferner wird auch der Dekoder IC 10 über Pin 5 freigegeben.

An einem der acht Ausgänge des Dekoders liegt nun ein 0-Signal, je nachdem, welche Adresse an A13 bis A15 liegt. Damit wird einer der Bausteine IC 1 bis IC 7 aktiviert. Als Bausteine können ohne Schaltungsänderung wahlweise EPROMs vom Typ 2764 oder RAMs vom Typ HM 6264 (oder äquivalente 8 K x 8-Speicher) eingesetzt werden. Diese Bausteine sind trotz des unterschiedlichen Inhalts so konstruiert, daß man die gleiche Fassung verwenden kann.

Im Prinzip kann man auch Bausteine des Typs 6116 (2 K x 8) einsetzen, dann muß man jedoch an der Brücke JMP1 eine neue Einstellung vornehmen, denn die Bausteine 6116 besitzen nur 24 Anschlußbeine und nicht 28, wie der 2764 oder HM6264. Anstelle der Adresse A11 wird dann durch die Brücke JMP2 das Signal -WR an das Pin 23 (bei 28poliger Zählweise) geführt.

Allerdings bleibt bei der Verwendung der 6116-Bausteine eine Adreßlücke, denn der Baustein wird nun viermal adressiert, bevor der nächste ausgewählt wird. Achtung: der Einsatz von 6116 ist nur für Sonderfälle vorgesehen, normalerweise werden die 8 K x 8-Bausteine eingesetzt und die Brücke JMP1 bleibt so, wie sie im Layout eingestellt ist.

Abb. 7.5.2 zeigt die Löt-, *Abb. 7.5.3* die Bestückungsseite. Der Bestückungsplan ist in *Abb. 7.5.4* abgedruckt.

Aufbau und Test

1. Löten Sie alle passiven Bauteile und alle Fassungen ein.
2. Schalten Sie die Versorgungsspannung ein und kontrollieren Sie, ob ggf. ein Kurzschluß vorliegt.
3. Ausschalten und IC 9, IC 10, IC 11 und IC 12 einsetzen.
4. Alle vier Brücken (JMP2) einsetzen.
5. Test: Zusammen mit der CPU68K auf den Bus setzen. Es stecken noch keine EPROMs in den Fassungen.

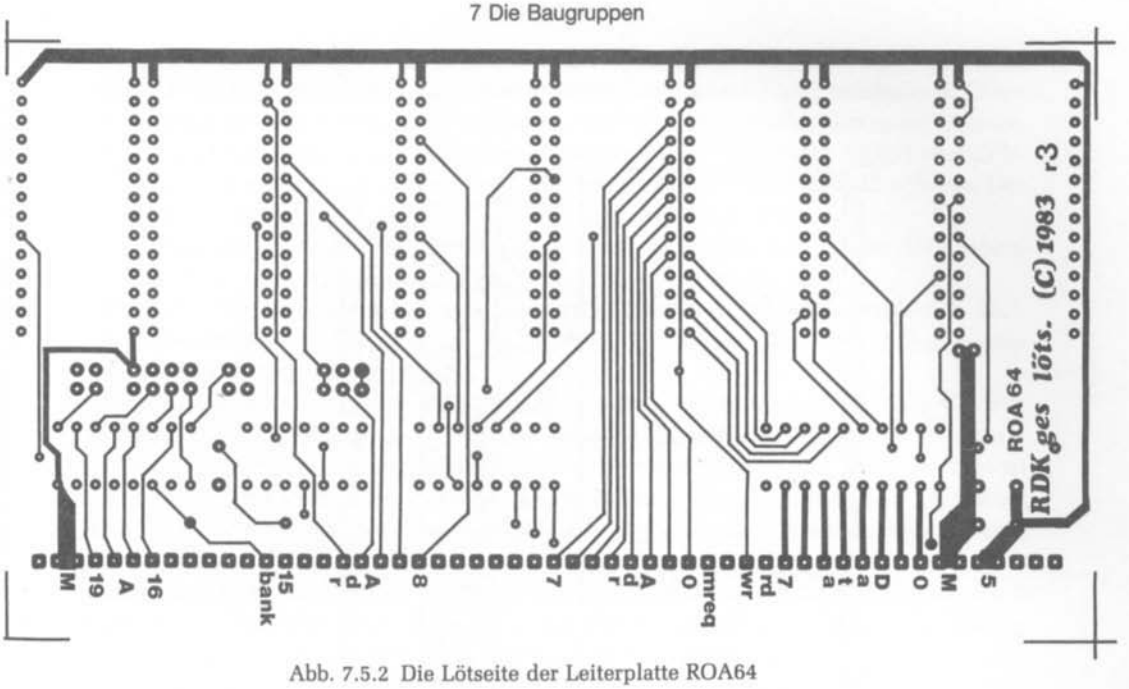


Abb. 7.5.2 Die Lötseite der Leiterplatte ROA64

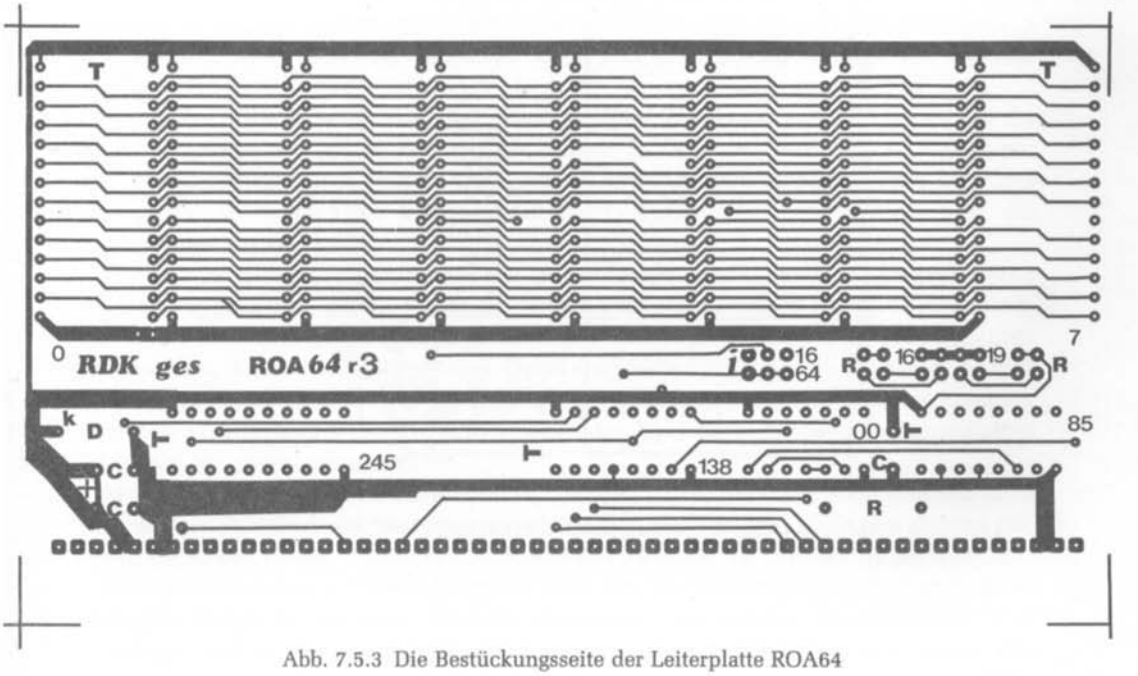


Abb. 7.5.3 Die Bestückungsseite der Leiterplatte ROA64

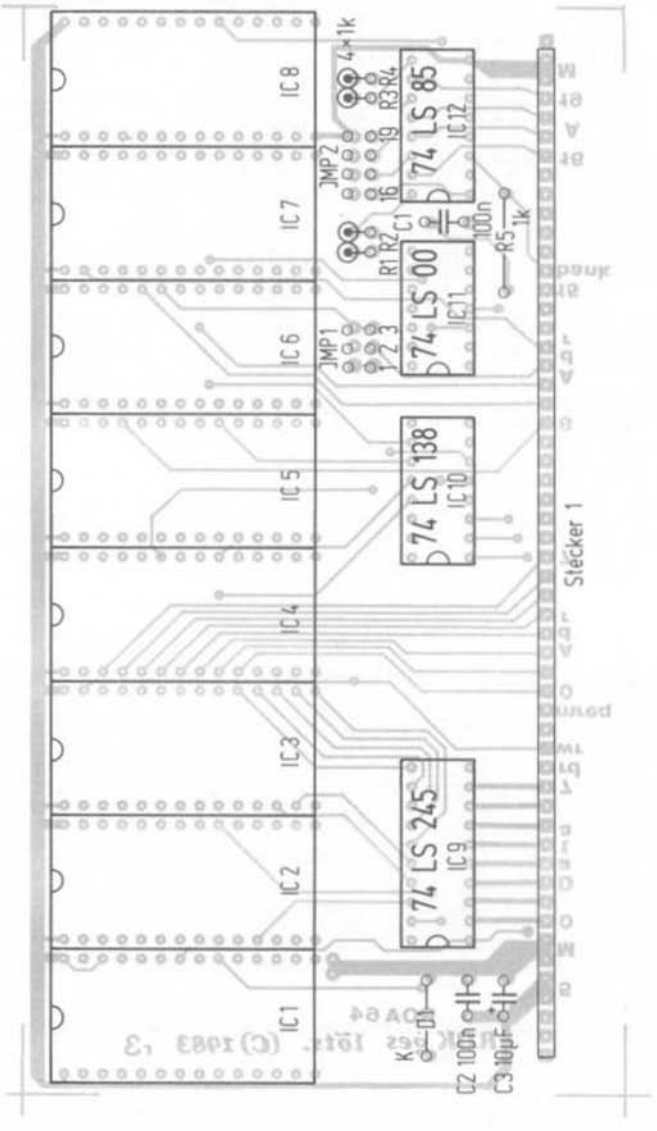


Abb. 7.5.4 Der Bestückungsplan der Baugruppe ROA64

Nach dem Einschalten der CPU müssen an Pin 19 des ICs 74LS245 und an Pin 15 des ICs 74LS138 mindestens acht (beim 68000 vier) kurze Pulse erscheinen. Diese Pulse sind nur mit einem Skop zu sehen, und auch damit nur schwer zu triggern. Sollte das Signal gar nicht erfaßbar sein, so kann man den Timer NE555 der CPU-Baugruppe herausziehen und an Pin 3 des Sockels einen 100-kHz-Takt anlegen. Der Test kann auch ausgelassen werden, wenn er zu schwierig ist.

6. Einsetzen der EPROMs. Die EPROMs des Grundprogramms kommen in die Fassungen für IC 1, IC 2, IC 3 und IC 4, das RAM in die Fassung für IC 5.
Wenn die GDP64-Baugruppe fertig ist, stecken Sie sie dazu, und es muß sich nach dem Einschalten das Grundprogramm melden. Tut es das nicht, helfen Ihnen vielleicht die Pulsdiagramme weiter.

Abb. 7.5.5 zeigt den Vorgang nach dem Einschalten. Dabei sind die Signale -CS (Pin 19, IC9) und die Ausgänge des Dekoders 74LS138 interessant. Wenn das Programm arbeitet, muß sich der Anfang so darstellen wie in der Abbbildung. Dabei kommt es nicht so genau auf Details an, vielmehr auf die Reihenfolge am Anfang. Das Diagramm gilt für die Version 4.3 des Grundprogramms. Bei anderen Versionen können sich Abweichungen ergeben.

Abb. 7.5.6 zeigt den Anfang nochmals in einer gedehnten Darstellung. Die ersten 8 Pulse werden immer erzeugt, auch wenn die Dateninformationen noch nicht ankommen; die folgenden Pulse sind dann bereits vom EPROM-Inhalt abhängig. Nach einer Weile muß sich ein stabiler Zustand ergeben; *Abb. 7.5.7* zeigt den Ablauf des immer wiederkehrenen Pulsverlaufes. Auch dieser Ablauf läßt sich mit einem Skop nachweisen. Die vier aufeinanderfolgenden Pulse weisen auf einen Stack-Zugriff hin. Sie sind an dieser Stelle nur in Systemen zu sehen, die ausschließlich mit einem RAM-Baustein arbeiten. In anderen Systemen treten diese Pulse an einem anderen Pin des Dekoders in Erscheinung. Man sollte im Fehlerfall auch mal die anderen Signale am Dekoder nachmessen.

Wenn Sie diese Pulse in ähnlicher Form auf dem Skop nachweisen können, ist die ROA-Baugruppe mit großer Wahrscheinlichkeit in Ordnung. Falls Sie sich jetzt mit einer Fehlersuche beschäftigen, weil vorher die GDP64-Baugruppe keine Meldung gezeigt hat, so müssen Sie im Abschnitt der GDP-Baugruppe weiterlesen, bzw. die Fehlersuche fortsetzen.

Die hier gemachten Angaben sind nur im Fehlerfall interessant. Normalerweise muß die Baugruppe gleich laufen. Sollte sie dennoch nicht auf Anhieb gehen, so kontrollieren Sie zuerst alle Lötstellen, dies ist auch hier, wie bei allen neu aufgebauten Baugruppen, der Fehler Nummer eins. Dabei sind kalte Lötstellen, nicht angelötete Pins und Kurzschlüsse durch Lötzinnbatzen an erster Stelle zu nennen. Seltener kommen Ätzfehler vor, die aber leider nur schwer zu finden sind. Bei selbstgeätzten Leiterplatten ist darauf besonders zu achten.

Wenn Sie schon einen funktionierenden NDR-KLEIN-Computer haben und nur noch eine weitere ROA64 dazubauen, so vereinfacht sich der Test sehr stark. Das gleiche gilt, wenn Ihnen bei einem Freund ein NDR-KLEIN-Computer zur Verfügung steht.

Dann können Sie einen RAM-Baustein, bzw. den Steckplatz dafür nach dem anderen austesten, indem Sie zunächst einen Baustein nach dem anderen, beginnend bei IC 1, einsetzen, und im Menue des Grundprogramms „Speicherbereiche" auswählen. Das

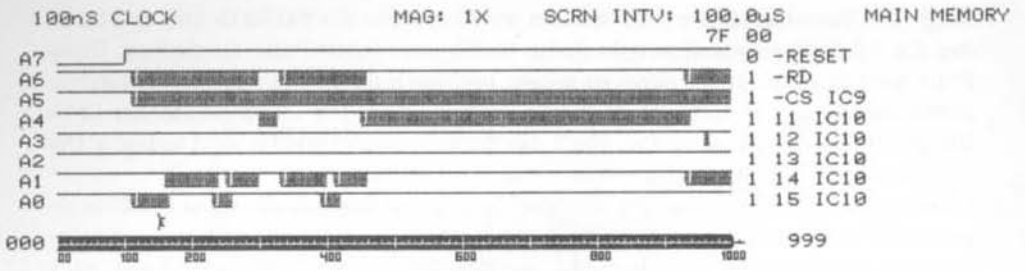


Abb. 7.5.5 Die Signale nach dem Einschalten der CPU

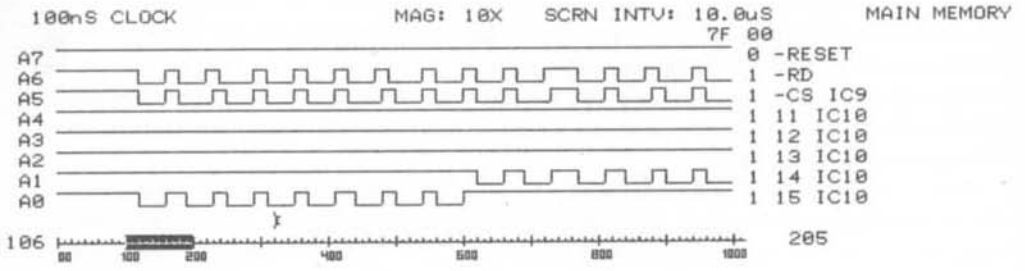


Abb. 7.5.6 Die ersten 10 µs nach dem Einschalten

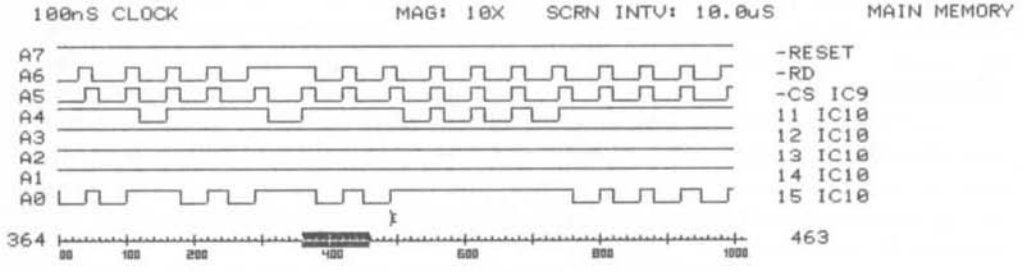


Abb. 7.5.7 Das Diagramm, wenn das Grundprogramm arbeitet

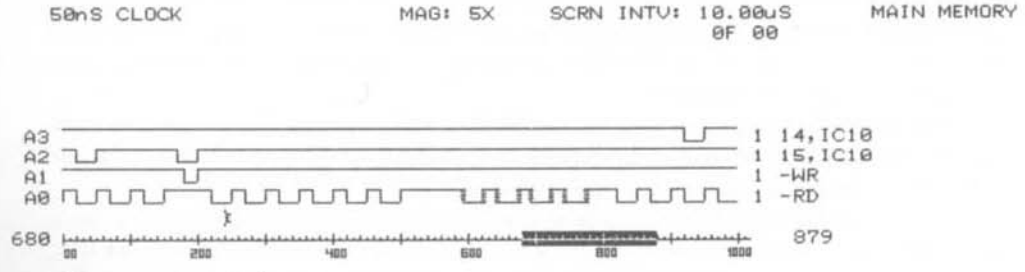


Abb. 7.5.8 Die Pulse beim Testprogramm

RAM muß dann dort angezeigt werden. Achtung: wählen Sie immer einen freien Adreßbereich für die neue ROA64-Baugruppe (JMP2)!

Abschließend noch ein kleines Testprogramm, mit dem Sie Fehler beim Dekodieren entdecken können.

```
START:
LEA $10000,A0        * hier Baugruppe-Adresse eintragen
CLR.L D0             * Schleifenwert
MOVE #8-1,D3         * 8 Dekoderausgänge

SCHLEIFE:
CLR.B 0(A0,D0.L)     * Lesen und Schreiben
ADD.L #$2000,D0      * nächster Ausgang
DBRA D3,SCHLEIFE     * 8 Mal
BRA START            * und zurueck.
```

In der ersten Programmzeile nach „START" wird mit dem „LEA"-Befehl die Startadresse definiert. $10000 ist hier nur als Beispiel gewählt. Hier muß die Anfangs-Adresse der Baugruppe eingesetzt werden. Das Programm sollte natürlich nicht in der fraglichen ROA64-Baugruppe abgelegt werden.

Nach dem Übersetzen mit dem Assembler und dem Start bei „START" werden die einzelnen Dekoderausgänge nacheinander angesprochen. *Abb. 7.5.8* zeigt einen Ausschnitt aus dem Timing. Zunächst erscheinen zwei Pulse am Pin 15 des Dekoderausganges. Der erste Puls kommt von einem Lesevorgang (das ist beim CLR-Befehl immer so!), dann ein weiterer Puls mit einem Schreibvorgang, dort wird der Wert in eine ggf. vorhandene Speicherzelle geschrieben. Nach fast 10 µs (je nach Warte-Zyklen auf der CPU) erfolgt der nächste Zugriff, diesmal über den Dekoderanschluß Pin 14. So werden alle 8 Ausgänge nacheinander angesprochen. Man kann durch den Lesepuls mit dem Skop auch den Datenbus kontrollieren. Falls sich im fraglichen Bereich RAM-Bausteine befinden, muß er zu diesem Zeitpunkt 0-Signale führen. Sind EPROMs eingesetzt, zeigt er nur dann 0-Signale, wenn ein Schreibvorgang stattfindet. So kann man auch den Speicher grob testen.

Der Trigger des Skopes wird dazu am besten mit dem Dekoderausgang verbunden, an den Y-Eingang kann man dann das Datensignal legen und es darstellen. Dazu muß man sich ggf. die zweite Stelle des Pulses markieren.

Folgende Änderung ist zum Test der Adreßleitungen geeignet: Im obengenannten Programm müssen wir zunächst die Anzahl der Schleifendurchläufe erhöhen. Statt #8-1 schreiben wir z.B. #$2000-1, und wo wir bisher den Wert #$2000 addierten, verwenden wir jetzt #1.

Abschließend noch ein Hinweis: sind in einer ROA64-Baugruppe nur RAMs eingesetzt, so kann man sie im Prinzip mit einer Batterie-Pufferung versehen. Dazu muß man die Versorgung der RAMs von der restlichen Schaltung trennen. Außer den Umgestaltungen bei der Versorgungsspannung ist auch noch eine Vorkehrung für die -WR-Leitung zu treffen. Nach dem Abschalten erscheint ja auf dieser Leitung ein 0-Signal (=Schreiben). Wird ein Baustein dann auch noch kurzzeitig selektiert, so gehen Daten verloren. Abhilfe schafft eine von der Batterie mitversorgte Schaltung, die die Schreibleitung des

Busses vom Speicher trennt (z.B. ein einfacher nichtinvertierender Baustein). Um die Stromaufnahme dieser Schaltung gering zu halten, sollte man auf CMOS-Bausteine zurückgreifen.

7.6 Die dynamische Speicherkarte

Eine Baugruppe, die bis zu 256 KByte RAM-Speicher aufnehmen kann, soll in diesem Abschnitt aufgebaut werden. Dabei kann man zwischen zwei Speichertypen wählen, dem 4164 mit 64 K x 1 Bits und dem 41256 mit 256 K x 1 Bits.

Die Baugruppe hat gegenüber der ROA64-Baugruppe mit statischen Speichern aber auch einen Nachteil. So benötigt die Baugruppe viele Wartezyklen, um arbeiten zu können. Dies kommt daher, daß die Schaltung der dynamischen Speicherbaugruppe möglichst einfach gehalten wurde, ohne spezielle integrierte Bausteine zu enthalten. Sie läuft dafür aber mit verschiedenen Prozessoren ohne Probleme und ist leicht nachzubauen.

Materialliste:

IC1,IC2,IC5,IC6,IC7 74LS153, Multiplexer
IC3,IC4 74LS161, Zähler
IC8 74LS04, Inverter
IC9 74LS74, D-Flip-Flop
IC10..IC17 4164 oder 41256 (oder baugleiche), Speicher
IC18 74LS02, Nor-Glieder
IC19 74LS164, Schieberegister
IC20,IC22 74LS244, unidirektionale Bustreiber
IC21 74LS32, Oder-Glieder
IC23 74LS05, Inverter mit offenem Kollektor
IC24 74LS85, Vergleicher
Q1 SG31, 24.000 MHz, intergrierter
Quarzoszillator
2× 20polige Fassung
16× 16polige Fassung
6× 14polige Fassung
R1, R2, R3, R4, R5, R6, R7, R10, R11, R13, R14, R15 33 Ω 1/8 W
R8, R9, R12, R16, R17, R18, R19, R20, R21, R22, R23,R24, R25, R26 1 kΩ 1/8 W
14× 100 nF
4× 10 µF, 16 V
1× Leiterplatte RAM64/256
1× Stiftleiste, gewinkelt (1× 36polig + 1× 18polig)
1× doppelreihe Stiftleiste 4polig, gerade (JMP3)
1× einreihige Stiftleiste 2polig (JMP3)
4× Shuntstecker

Kenndaten:

Spannung: +5 V, Stromaufnahme, voll bestückt mit 256 KByte: 400 mA

Abb. 7.6.1 zeigt den kompletten Schaltplan.

Die Adresse der Baugruppe wird mit JMP3 eingestellt; der Vergleicher IC 24 spielt bei der Festlegung der Adressen eine wesentliche Rolle. Nachdem die Baugruppe mit den beiden bereits genannten Bausteintypen betrieben werden kann, gibt es verschiedene Adreßbereiche. Ist die Brücke JMP1 eingesetzt, wird die Leitung A8' an die Speicherbausteine geführt; der 256-K-Betrieb ist eingestellt. In diesem Zustand finden auch die Adressen A16 und A17 Verwendung. Am Vergleicher sind sie dann eigentlich überflüssig. Daher muß man beim Betrieb mit den 256-K-Bausteinen auch die Brücken A und B am JMP3 einsetzen. Damit werden Pin 9 mit Pin 10 und Pin 11 mit Pin 12 am Vergleicher IC 24 verbunden. Der Wert an dieser Adresse spielt dann für die Dekodierung keine Rolle mehr.

Die Brücken bei A19 und A18 wählen also bei den 256-K-Bausteinen den Adreßbereich aus:

A19	A18	Bereich
gesteckt	gesteckt	00000-3FFFF
gesteckt	offen	40000-7FFFF
offen	gesteckt	80000-BFFFF
offen	offen	C0000-FFFFF

Achtung: bei der letzten Einstellung ist der Bereich F0000 bis FFFFF beim 68008 nicht verwendbar, da dort der IO-Bereich liegt.

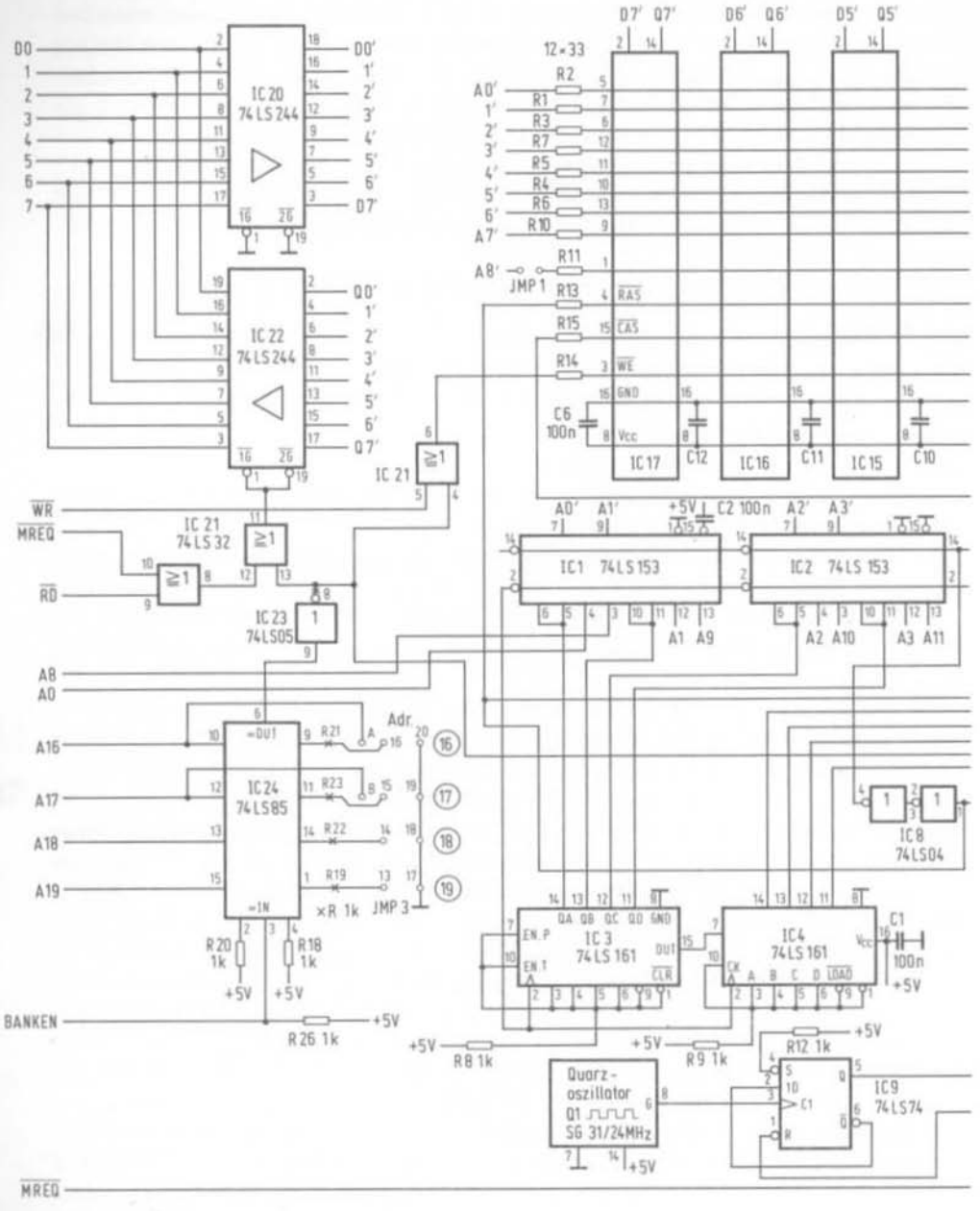
Bei der Verwendung von 64-K-Bausteinen werden alle vier Brücken (A16 bis A19) verwendet. Der Speicherbereich kann dann auf eine der 16 Adressen gelegt werden.

Zu beachten ist dabei, daß die 16te Adresse (F0000 bis FFFFF) beim 68008 wegen der IO-Adressierung nicht sinnvoll ist. Die Brücken A und B des JMP3, sowie JMP1 bleiben dann offen.

Mit der Brücke JMP2 werden die Wartezyklen eingestellt. Normalerweise ist diese Brücke bereits im Layout auf Position 3 festgelegt. Dies entspricht der Einstellung für das Z80-System. Bei der Verwendung des 68008 kann die Brücke auch bei Position 1 eingesetzt werden. Dadurch wird das -WAIT-Signal früher zurückgesetzt. Da der Prozessor nach dieser Zurücknahme noch Zeit für die Erkennung dieser Änderung benötigt, reicht der Zyklus aus. Am besten läßt sich experimentell ermitteln, wo man die Brücke noch hinsetzen darf, indem man das Menue „Speicherbereiche" heranzieht. Für den ersten Start sollte sich die Brücke auf jeden Fall auf Position 3 befinden.

Die Bausteine IC 1 bis IC 7 haben die Aufgabe, die unterschiedlichen Adressen zu multiplexen. Bei den dynamischen Speichern werden die 16 bzw. 18 Adressen nämlich nur über 8 bzw. 9 Leitungen zeitlich hintereinander übertragen. Zur Koordination dieses Vorganges gibt es ein Signal -RAS, das mit seiner fallenden Flanke angibt, wann der erste Adreßteil anliegt, und außerdem ein Signal -CAS, das ebenfalls mit der fallenden Flanke die zweite Adreßhälfte übernimmt.

Dynamische Speicher würden nach einer bestimmten Zeit ihren Inhalt verlieren, würde nicht rechtzeitig ein Refresh erfolgen. Im Klartext bedeutet das, alle 2 ms (oder 4 ms je nach Baustein) müssen 128 bzw 256 verschiedene Reihen angesprochen werden. Dazu dienen die Zähler IC 3 und IC 4. Sie erzeugen eine 8-Bit-Adresse, die bei jedem Speicherzyklus erhöht wird. Die Reihen spricht man mit dem -RAS-Signal an, das eigens dazu erzeugt wird.


IC 20
74 LS 244
IC 22
74 LS 244
IC 21
74 LS 32
IC 23
74LS05
IC 24
74LS85
IC 1 74LS 153
IC 2 74LS 153
IC 3
74 LS 161
IC4
74 LS 161
IC 17
IC 16
IC 15
IC 8
74LS04
IC 9
74LS74
Quarz-
oszillator
Q1
SG 31/24MHz
JMP 1
JMP 3
R 1k
R2
R1
R3
R7
R5
R4
R6
R10
R11
R13
R15
R14
R12 1k
R8 1k
R9 1k
R18
1k
R20
1k
R21
R22
R23
R19
R26 1k
C6
100n
C12
C11
C10
C2 100n
C1
100n
+5V
BANKEN
MREQ
WR
RD
Adr.

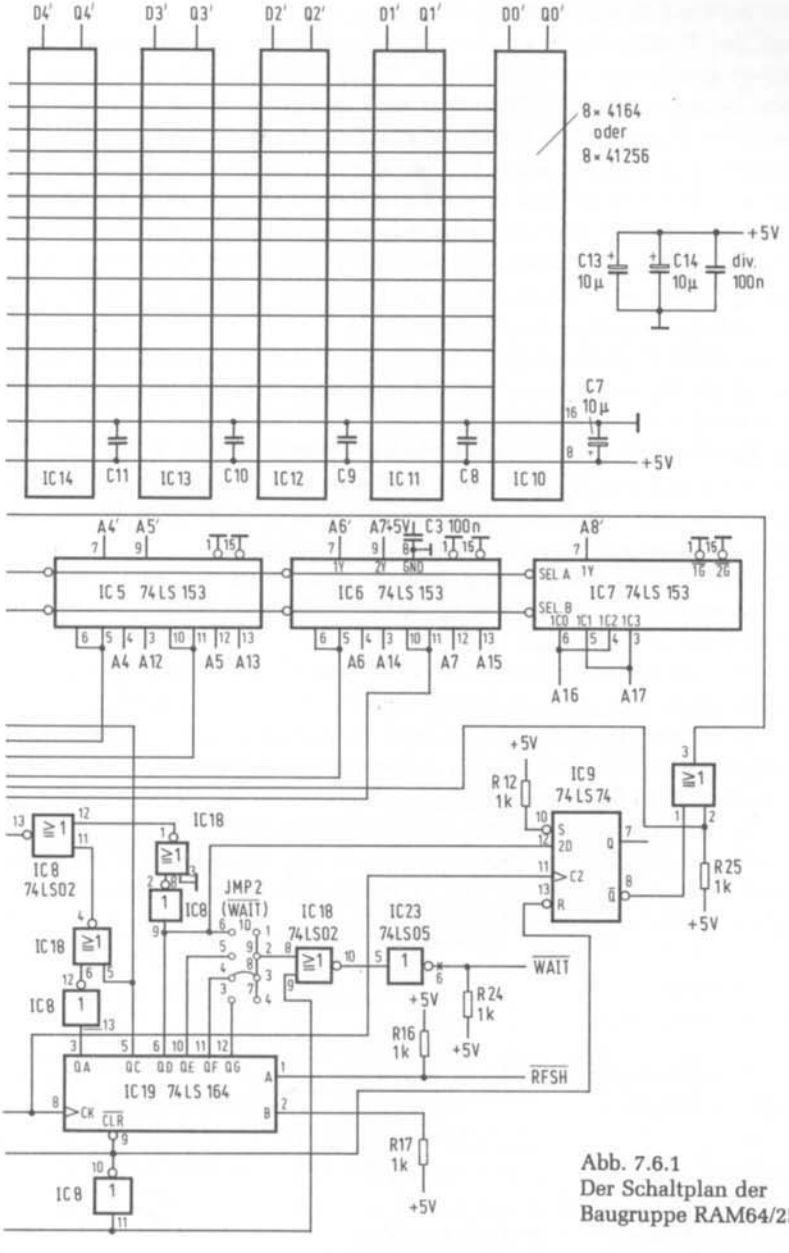


Abb. 7.6.1
Der Schaltplan der Baugruppe RAM64/256

Der Refresh wird beim NDR-KLEIN-Computer bei jedem Speicherzugriff ausgelöst; auch dann, wenn die Baugruppe nicht adressiert wird. Damit ist garantiert, daß die Bedingung, alle 2 ms ein Refresh, eingehalten wird.

Abb. 7.6.2 zeigt das Timing-Diagramm dazu. Oben sehen Sie das -RD und -MREQ-Signal. Hier beginnt gerade ein Lesezugriff. Die -WAIT-Leitung wird aktiv, denn die Speicherbausteine benötigen für ihre Reaktion eine gewisse Zeit. Dann wird -RAS erzeugt, dabei liegt das Signal SEL B (Select B, Pin 2 an den ICs 74LS153) auf 0. Der aktuelle Zählerinhalt der ICs 3 und 4 wird als Adresse an den Multiplexern angelegt. Sie wird in die Speicher übernommen und bewirkt den Refresh. Damit wäre die Aufgabe eigentlich erfüllt. Nun ist unsere Speicherbaugruppe leider nicht so intelligent, daß sie den restlichen Mechanismus nicht mehr ablaufen läßt. Es folgt noch ein zweites -RAS-Signal, und dann wird erst der -WAIT-Ausgang freigegeben. Damit wird leider auch jeder Zugriff von anderen Speichern verlangsamt.

Wie sieht das bei einem Zugriff auf die Baugruppe aus? *Abb. 7.6.3* zeigt das Timing. Der Ablauf ist praktisch der gleiche, nur daß hier nach dem zweiten -RAS-Signal auch ein -CAS-Signal erzeugt wird.

Übrigens liegt natürlich SEL B (Select B) beim zweiten Zugriff auf 1, um die Prozessor-Adressen auf den Bus zu legen. Die Umschaltung zwischen den niederwertigen und den höherwertigen Adressen geschieht mit SEL A (Select A, Pin 14 am 74LS153). Das SEL A-Signal wird kurz vor dem -CAS umgeschaltet.

Abb. 7.6.4 zeigt den Vorgang bei einem Schreib-Zugriff. Dabei wird vom Prozessor anstelle des -RD-Signals ein -WR-Signal erzeugt, das über die Oder-Verknüpfung (IC 21, Pin 4,5,6) an die Leitung -WE gelangt.

Eine Besonderheit beim 68008 stellt der TAS-Befehl dar. Er führt einen sogenannten Read-Modify-Write-Zyklus aus. Dies bedeutet, es wird bei einem Zugriff ein Wert gelesen und ein anderer zurückgeschrieben. Obwohl dieser Befehl nur für Mehrfachprozessor-Anwendungen gedacht ist, tritt er auch in mancher Anwendersoftware auf. *Abb. 7.6.5* zeigt das Zeitdiagramm für diesen Befehl.

Zu beachten ist in diesem Zusammenhang, daß ältere Baugruppen diesen Befehl noch nicht ausführen konnten. Um sie umzurüsten, muß man den Pin 3 des IC 18 an Masse legen und die zum Ausgang des IC 19, Pin 11 führende Leiterbahn auftrennen. In unserem Schaltplan ist dies bereits geschehen.

Ein kleines Testprogramm:
1. Für einen Test mit dem Skop:

```
START:
        TAS.B $40000 * hier Adresse eintragen
        BRA.S START
```

2. Zum Funktionstest:

```
START:
        CLR.B $40000      * Byte loeschen
        TAS.B $40000      * Setzt Bit 7 auf 1
        MOVE.B $40000,D0  * Ergebnis im Einzelschritt in D0
        RTS               * erkennbar
```

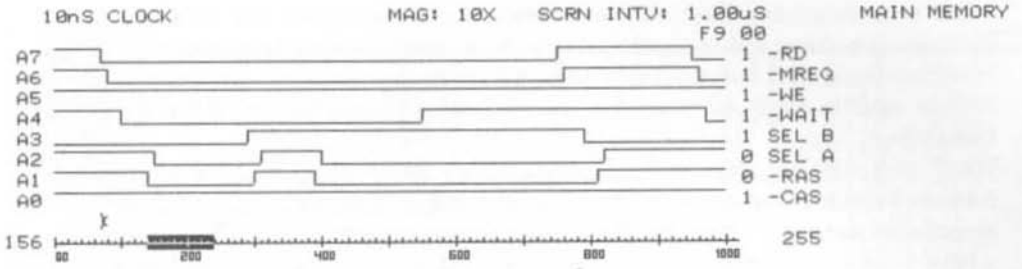


Abb. 7.6.2 Die Signale, wenn kein Zugriff erfolgt

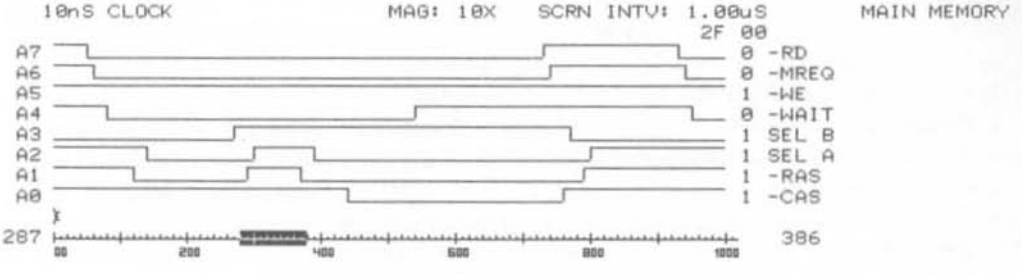


Abb. 7.6.3 Es wird von der Baugruppe gelesen

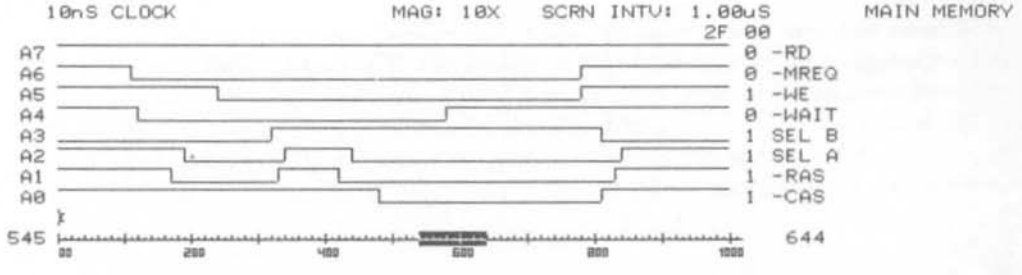


Abb. 7.6.4 Ein Wert wird geschrieben

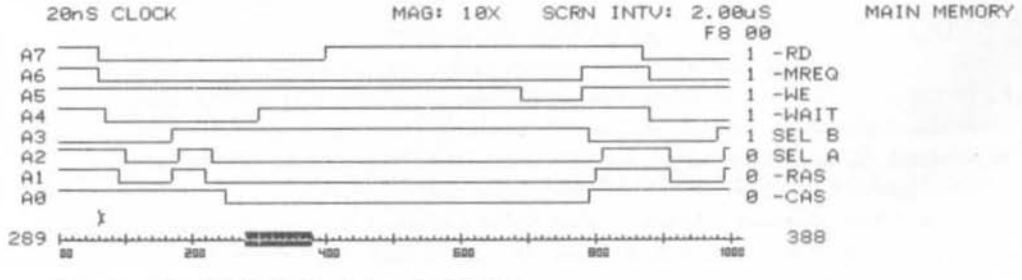


Abb. 7.6.5 Der TAS-Befehl mit dem RMW-Zyklus

Das erste Programm läuft endlos, es dient im Wesentlichen der Erzeugung eines Timings, das dann mit dem Skop beobachtet werden kann. Das zweite testet die Funktion. Dazu führt man das Programm im Einzelschritt aus, beim Erreichen des RTS-Befehls muß in D0.B der Wert $80 stehen (die restlichen Bits 8 bis 31 sind ohne Bedeutung).
Wenn sich die dynamische Speicherbaugruppe nicht über die jetzt eingetragene Adresse $40000 ansprechen läßt, so muß hier natürlich der von Ihnen gewählte Wert eingetragen werden.

Abb. 7.6.6 zeigt die Lötseite des Layouts und *Abb. 7.6.7* die Bestückungsseite. In *Abb. 7.6.8* ist der Bestückungsplan zu sehen.

Aufbau und Test:

1. Einlöten aller passiven Bauteile und aller Fassungen.
2. Spannung einschalten und auf Kurzschluß prüfen.
3. Spannung ausschalten. Alle TTL-Bausteine einstecken und den Quarzoszillator einlöten. Auf dem Oszillator markiert ein Punkt die Richtung der Nase, der Punkt ist Pin 1. Löten Sie jedoch nur sehr kurz, um den Oszillator nicht zu beschädigen.
4. Spannung einschalten. Dabei wird der NDR-KLEIN-Computer mit dem Grundprogramm in einer ROA64 als Testgerät verwendet. Messen Sie erst mal den Takt, er beträgt 24 MHz. Ein Prüfstift genügt zur groben Kontrolle.

   Setzen Sie nun die Brücke bei A19 ein. Die Baugruppe wird dann im Bereich $70000 bis $7FFFF adressiert. Dort sollte sich momentan keine ROA64-Baugruppe befinden.

   Wenn sich das Grundprogramm nicht mehr meldet, liegt ein Fehler, meist ein Kurzschluß, vor. Entfernen Sie dazu wieder alle ICs, und schalten Sie ein. Das Grundprogramm muß sich melden; wenn nein, kontrollieren Sie alle Lötstellen und Leiterbahnen.

   Ist diese Hürde genommen und meldet sich das Grundprogramm, kann man folgendes Testprogramm eingeben.

```
START:
        CLR.B $70000
        BRA.S START
```

Durch dieses Programm wird ein Lese- und Schreibversuch auf die Karte durchgeführt. Kontrollieren Sie nun verschiedene Punkte auf der Baugruppe. An Pin 1 und Pin 19 des IC 22 müssen ebenso einzelne Pulse auftauchen, wie an Pin 3 der Speicher (-WE). Außerdem müssen am -WAIT-Ausgang Pulse nachzuweisen sein. Es ist sinnvoll, direkt am jeweiligen Pin zu messen und nicht am Bus; dort findet ggf. eine Überlagerung mit dem DTACK-Mechanismus statt.

Die Adreßleitungen müssen diverse Pulse führen, achten Sie hier besonders auf Kurzschlüsse, die man an einem Pegel von ca. 1 V erkennt. Auf den Adreßleitungen an den Speichern dürfen nur eindeutige TTL-Pegel vorkommen.

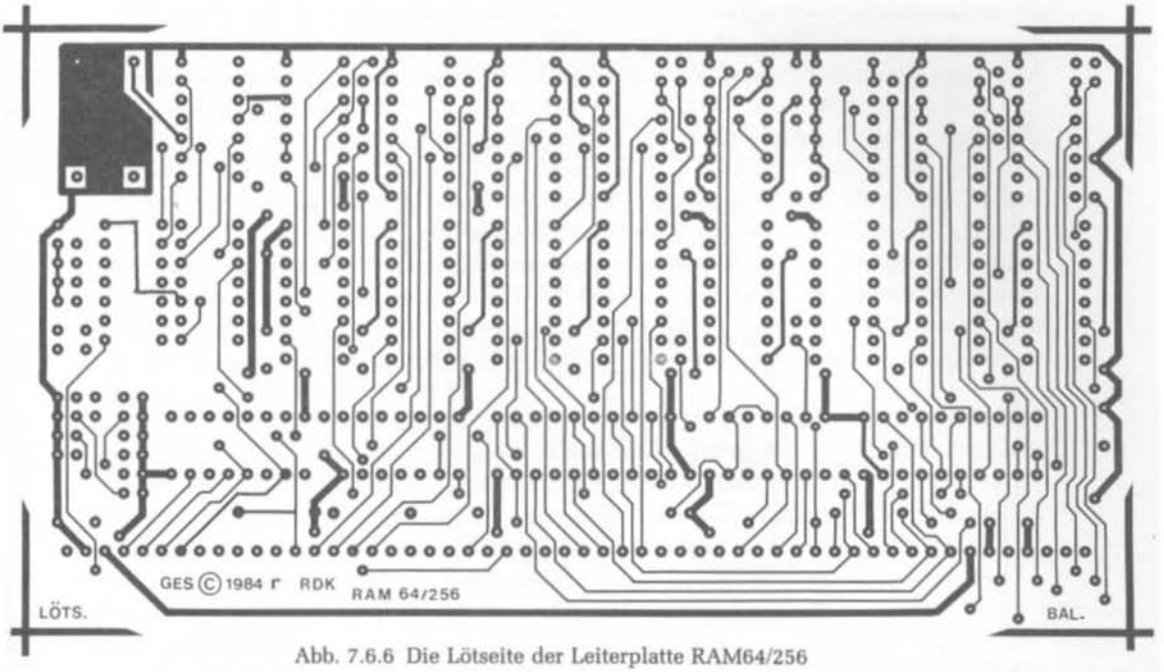


Abb. 7.6.6 Die Lötseite der Leiterplatte RAM64/256

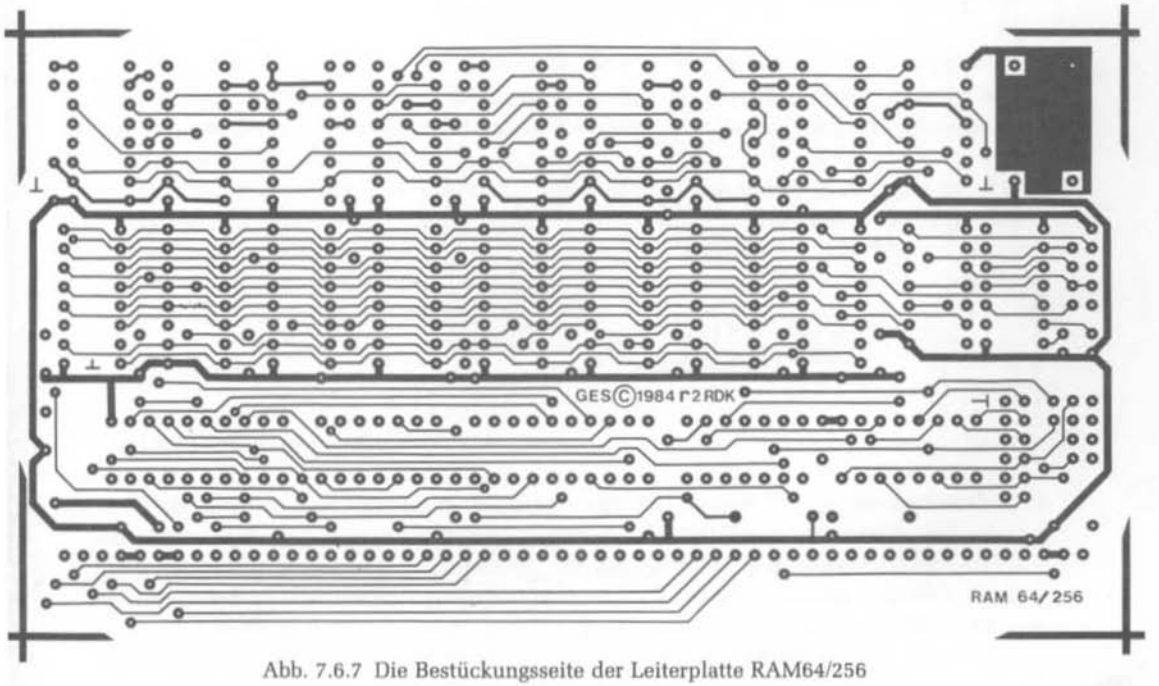


Abb. 7.6.7 Die Bestückungsseite der Leiterplatte RAM64/256

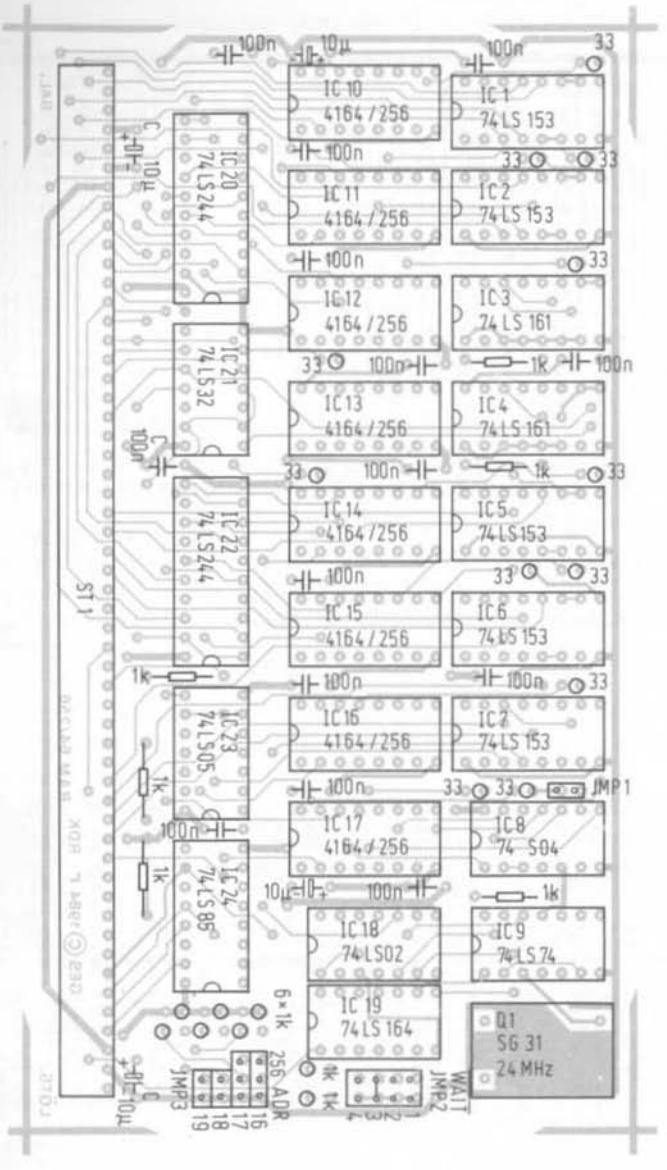


Abb. 7.6.8
Der Bestückungsplan der Baugruppe RAM64/256

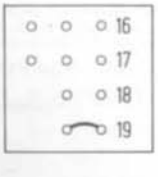


Abb. 7.6.9
Einstellung der Brücken für einen ersten Test

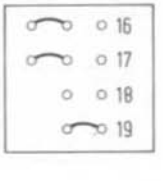


Abb. 7.6.10
So müssen die Brücken eingestellt werden, wenn man 256-K-Speicher-ICs einsetzt

5. Nun kann man es wagen, die Speicher einzusetzen; vorher die Spannung ausschalten. Achten Sie nochmals auf die Orientierung von Pin 1! Hinweis: bei den Speicherbausteinen liegt an Pin 16 Masse und an Pin 8 +5 V, also genau umgekehrt als bei TTL-Bausteinen.
6. Jetzt kommt es darauf an, ob Sie die 64 K x 1, oder die 256 K x 1 Speicher eingesetzt haben.

a) Bei den 64-K-Speichern brauchen Sie keine weiteren Brücken einzustellen. Die Adresse wird nur an JMP3 mit den vier rechten Brücken eingestellt (A19 bis A16). Lassen Sie zum Test die Brücke A19 eingesetzt. *Abb. 7.6.9* zeigt die Brückeneinstellung.

b) Bei den 256-K-Speichern müssen Sie JMP1 stecken, um die zusätzliche Adresse an die Speicher zu führen. Ferner stecken Sie die Brücken A und B bei JMP3, so daß A16 und A17 mit den korrespondierenden Eingängen des Vergleichers überbrückt werden. *Abb. 7.6.10* zeigt die Einstellung.

Im Menue „Speicherbereiche" muß nun im Fall a) der Bereich $70000 bis $7FFFF auftauchen und im Fall b) der Bereich $40000 bis $7FFFF. Sollte das nicht der Fall sein, müssen Sie sich anhand der Timing-Diagramme auf Fehlersuche machen.

Verbesserungsmöglichkeiten:

Eine Anregung an alle Leser, die die Leistung der Speicherkarte durch eigene Versuche verbessern wollen. Mit einem zusätzlichen Zähler kann man erreichen, daß der Refresh nicht so häufig durchgeführt wird. Dadurch wird die zeitliche Belastung durch Warte-Zyklen verringert.

Arbeitet die CPU nicht, wird auch kein Refresh ausgeführt. Dieser Umstand wird dann kritisch, wenn ein langer Reset oder ein lang andauernder Warte-Zyklus (durch eine andere Baugruppe) ausgelöst wird. Um hier Abhilfe zu schaffen, benötigt man ebenfalls zusätzliche Zähler.

Das Problem mit dem Reset läßt sich auch durch eine andere Änderung der Reset-Schaltung umgehen. Beim Spannungseinschalten soll dabei der Reset so lange anhalten wie jetzt; beim manuellen Reset jedoch kann man durch ein zusätzliches Monoflop verhindern, daß er länger als z.B. 100 µs dauert und damit den Speicherinhalt der dynamischen Speicher zerstört.

7.7 Die Baugruppe BANK/BOOT

Für den Betrieb von CP/M-68k benötigt man z.B. einen RAM-Speicherbereich ab Adresse 0. Dort muß aber nach dem Einschalten ein EPROM stehen, denn auf Adresse 0 bis 7 stehen die Einschaltinformationen. Die BANK/BOOT-Baugruppe erlaubt es, nach dem Einschalten im Adressenbereich 0 bis maximal $7FFF EPROMs zu haben, die sich später per Programm ausblenden lassen. Zum Betrieb mit dem CP/M-68k benötigt man nur ein EPROM 2764 (oder 2716) auf dieser Baugruppe, siehe Kapitel 6.

Materialliste:

IC 5 74LS245, bidirektionaler Bustreiber
IC 6 74LS273, 8fach-Zwischenspeicher
IC 7,IC 8 74LS85, Vergleicher
IC 9 74LS04, Inverter
IC 10 74LS138, 1-aus-8-Dekoder
IC 11 74LS32, Oder-Glieder
IC 1, IC 2 EPROMs 2764 (ggf. auch RAMs 6264)
IC 3, IC 4 RAMs 6116, oder nach Umstellen der
Brücke J1 auch EPROMs 2764 / oder
RAMs 6264 (8 K × 8).
Für CP/M-68k wird als IC 1 nur ein 2764 oder ersatzweise ein 2716 benötigt, da dort nur ein sehr kleines Programm (siehe Kapitel 6, BOOTR) untergebracht wird.
R1 330 Ω 1/8W
C1, C2 100 nF (nur auf dem Bestückungsplan)
C3 10 µF (nur auf dem Bestückungsplan)
LED1 Leuchtdiode rot, mit 3mm Durchmesser
1 × einreihige Stiftleiste, gewinkelt (1 × 36polig + 1 × 18polig)

Achtung: Die vier im Layout festgelegten Brücken bei J3 müssen für den 68008-Betrieb aufgetrennt werden!

Kenndaten:

Spannung: +5 V, Stromaufnahme bei voller Bestückung (2 x EPROMs + 2 × 6116 RAMs): 200 mA

Abb. 7.7.1 zeigt die Schaltung der BANK/BOOT-Baugruppe. Im Prinzip arbeitet sie genauso wie die ROA64-Baugruppe, nur daß hier noch eine Logik für die Ausblendung von anderen Speicherbänken hinzukommt.

Dazu wird das Signal BANKEN erzeugt. Immer wenn es auf 0 liegt, werden andere Speicherbänke ausgeblendet. Bei den zu steuernden Baugruppen ROA64 und RAM64/256 ist es jeweils ein Eingang. BANKEN wird immer dann auf 0 gelegt, wenn die BANK/BOOT-Baugruppe adressiert wird. Dabei muß A15 auf 0 liegen, dazu dient das IC 11, das A15 und den Ausgang von IC 6, Pin 12 miteinander oder-verknüpft. BANKEN liegt also immer dann auf 0, wenn der Ausgang Pin 12 des IC 6 und A15 auf 0 liegen. Damit erhält die Baugruppe die Adreßbereiche 0...7FFF, 10000...17FFF, 20000...27FFF usw.

IC 6 ist ein Zwischenspeicher. Er wird durch das -RESET-Signal auf 0 gebracht, so daß an allen Ausgängen des ICs ein 0-Signal liegt. Nach dem Einschalten wird die Baugruppe damit adressiert. Eine Leuchtdiode LED1 zeigt an, wann die Baugruppe aktiv ist. Der Dekoder IC 10 wird immer dann freigegeben (über Pin 4), wenn die Baugruppe aktiv ist, dabei muß auch noch das -MREQ-Signal anliegen.

Der Dekoder wählt dann eines der ICs 1 bis 4 aus. Wenn ein Zugriff erfolgt, wird das IC 5 über Pin 19 ebenfalls freigeschaltet.

Nun beinhaltet die Baugruppe auch noch einen IO-Port, nämlich das IC 6. Es besitzt eine IO-Adresse, die durch die beiden Vergleicher IC 7 und IC 8 bestimmt ist.

Die Adresse ist fest auf $FFFFFFC8 eingestellt. Wird auf diese Adresse geschrieben, so kann man den Inhalt des Zwischenspeichers IC 6 ändern. Wenn man z. B. den Wert $80 einschreibt (MOVE.B #$80,$FFFFFFC8), so wird die Baugruppe inaktiv, denn nun

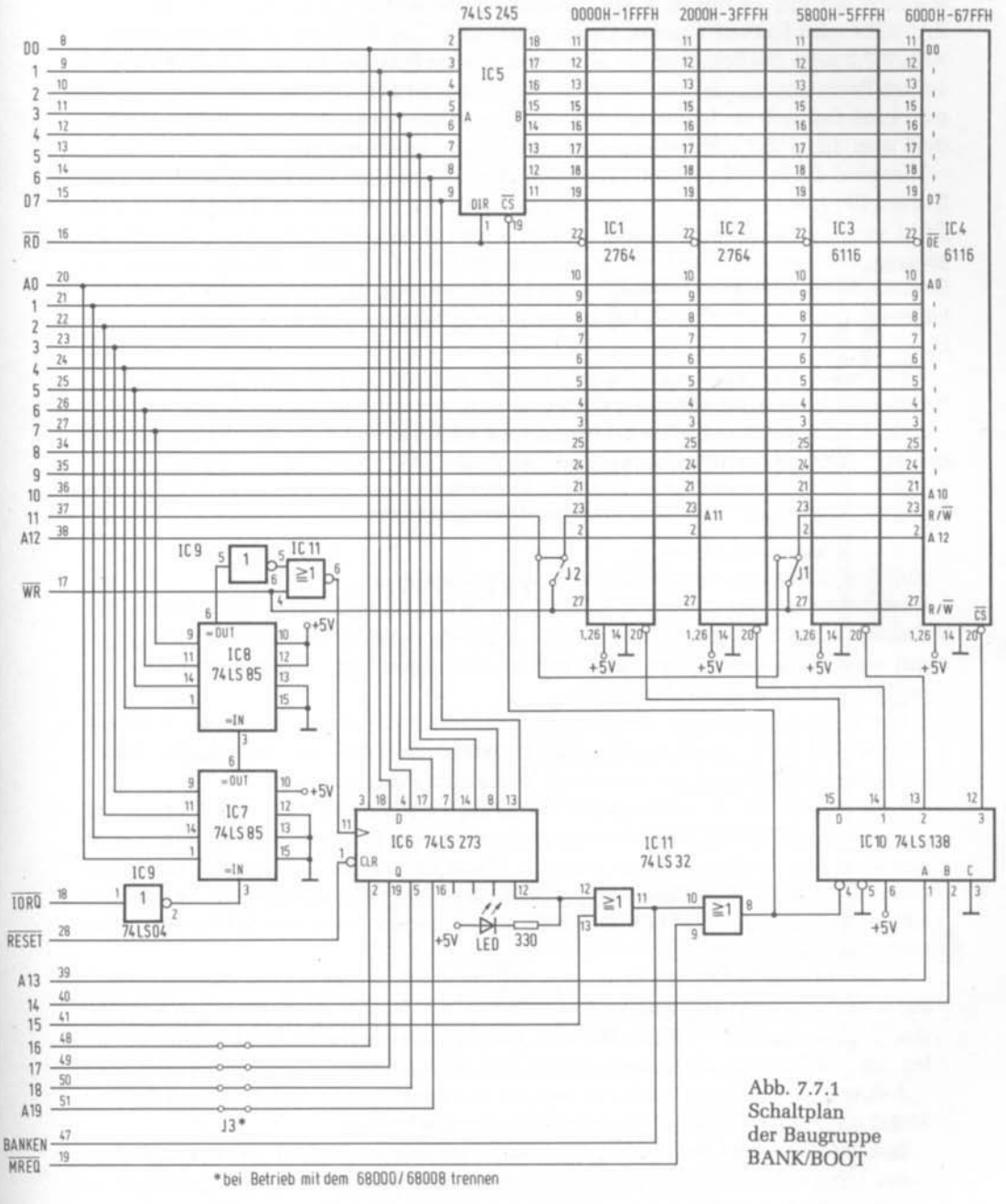


Abb. 7.7.1
Schaltplan
der Baugruppe
BANK/BOOT

*bei Betrieb mit dem 68000/68008 trennen

erscheint an Pin 12 des IC 6 ein 1-Signal, die Oder-Bedingung an IC 11 ist erfüllt, BANKEN wird fest auf 1 gelegt. Damit wird jeder Zugriff auf die Baugruppe gesperrt. *Abb. 7.7.2* zeigt die Belegung der einzelnen Bits am Port. Die Bits 0 bis 3 sind dabei für andere Prozessoren, wie den Z80, gedacht, die nur 64 KByte RAM adressieren können; man kann dadurch die fehlenden Adressen zusätzlich erzeugen. Beim 68000 und 68008 muß man daher die vier Leitungen bei der Brücke J3 auftrennen.

Hier sind die Brücken J1 und J2 so verdrahtet, daß sich bei den Steckplätzen IC 1 und IC 2 ein EPROM vom Typ 2764 befinden kann. An den Positionen IC 3 und IC 4 ist je ein RAM vom Typ 6116 vorgesehen. Dies ist auch für die Z80-Anwendung mit CP/M80 gedacht, für uns aber nicht weiter interessant. Beim CP/M-68k benötigt man nur ein EPROM in der Position IC 1. Dabei kann es sich wahlweise um ein 2764 oder ein 2716 handeln. Beide werden auch bei unveränderter Leiterbahnführung richtig angesprochen.

Wer die Karte für andere Zwecke verwenden will und so z. B. auch bei IC 3 und IC 4 RAMs oder EPROMs mit 8 K x 8 einsetzen will, der muß die Brücke J1 umstellen.

Abb. 7.7.3 zeigt die Lötseite der Baugruppe und *Abb. 7.7.4* die Bestückungsseite. In der *Abb. 7.7.5* ist der Bestückungsplan dargestellt.

Aufbau und Test:

1. Einlöten aller passiven Bauteile und aller IC-Fassungen.
2. Einschalten der Versorgungsspannung, und einen Kurzschlußtest druchführen.
3. Einsetzen der restlichen ICs sowie des EPROMs BOOTR. Wenn Sie die Baugruppe mit eigener Software betreiben, müssen Sie dieses jetzt einbauen und den Test entsprechend verändert durchführen.

   Die ROA64-Baugruppe soll beim ersten Test auf Adresse 0 bleiben. CPU68K, GDP64 und KEY sollen auch vorhanden sein, und ohne den Einsatz der BANK/BOOT-Baugruppe muß sich das Grundprogramm normal melden.

   Nun stecken Sie die BANK/BOOT-Baugruppe mit dem EPROM BOOTR dazu und schalten ein. Die LED muß anfangs leuchten und nach einer Weile ausgehen. Dann soll sich das Grundprogramm melden. Wenn es das tut, arbeitet die Baugruppe bereits. Wenn Sie nun auch noch bereits eine Speicherbaugruppe mit 64 oder 256 KByte besitzen (für CP/M-68k benötigt man mindestens 128 KByte!), können Sie den nächsten Test durchführen.

   Adressieren Sie die RAM-Baugruppe ab Adresse 0 und die ROA64-Baugruppe z. B. auf Adresse $E0000 (nur Brücke A16 steckt). Nach dem Einschalten muß sich wieder das Grundprogramm melden. Mit dem Menue „Speicherbereiche" können Sie prüfen, ob der Speicher ab Adresse 0 auch funktionsfähig ist.

   Achtung, das Programm BOOTR arbeitet nur dann korrekt, wenn sich auf Adresse $8000 ein kleiner, funktionierender RAM-Bereich befindet.

   BOOTR findet das Grundprogramm an verschiedenen Stellen im Hauptspeicher, daher konnte der erste Test auch ohne verschobenes Grundprogramm durchgeführt werden. Für diesen Test ist die Version 4.3 des Grundprogramms erforderlich.

   Lesen Sie auch den entsprechenden Abschnitt im Kapitel 6 durch, falls es Probleme gibt.

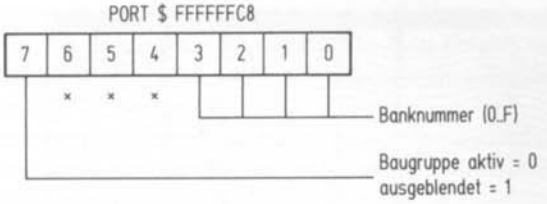


Abb. 7.7.2
Die Belegung des Ports $FFFFFFC8

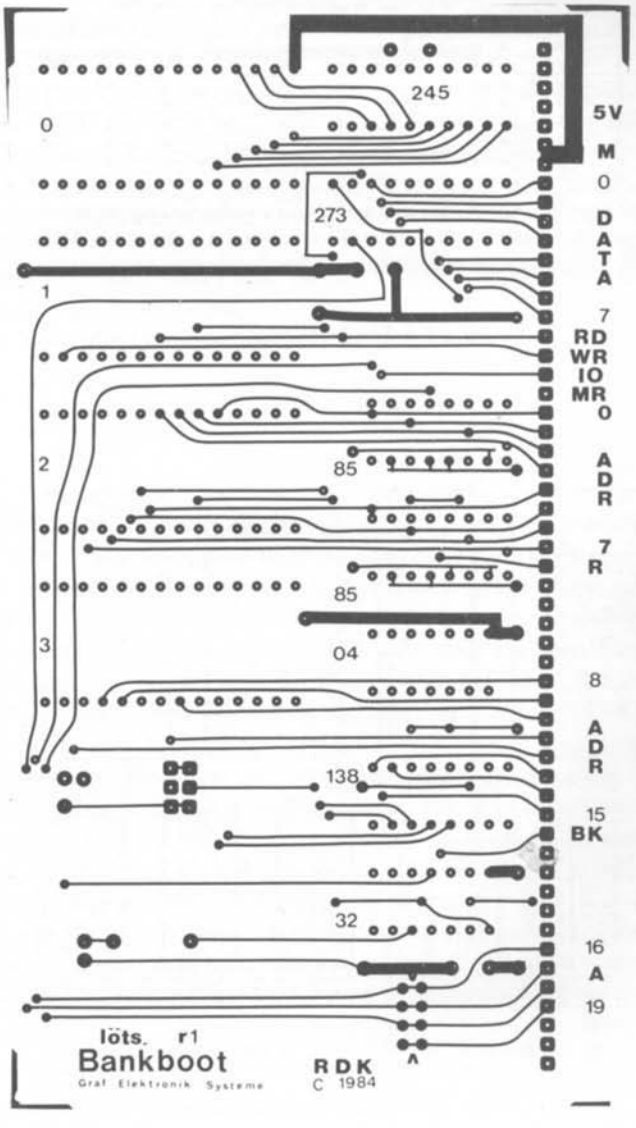


Abb. 7.7.3
Die Lötseite der Leiterplatte BANK/BOOT

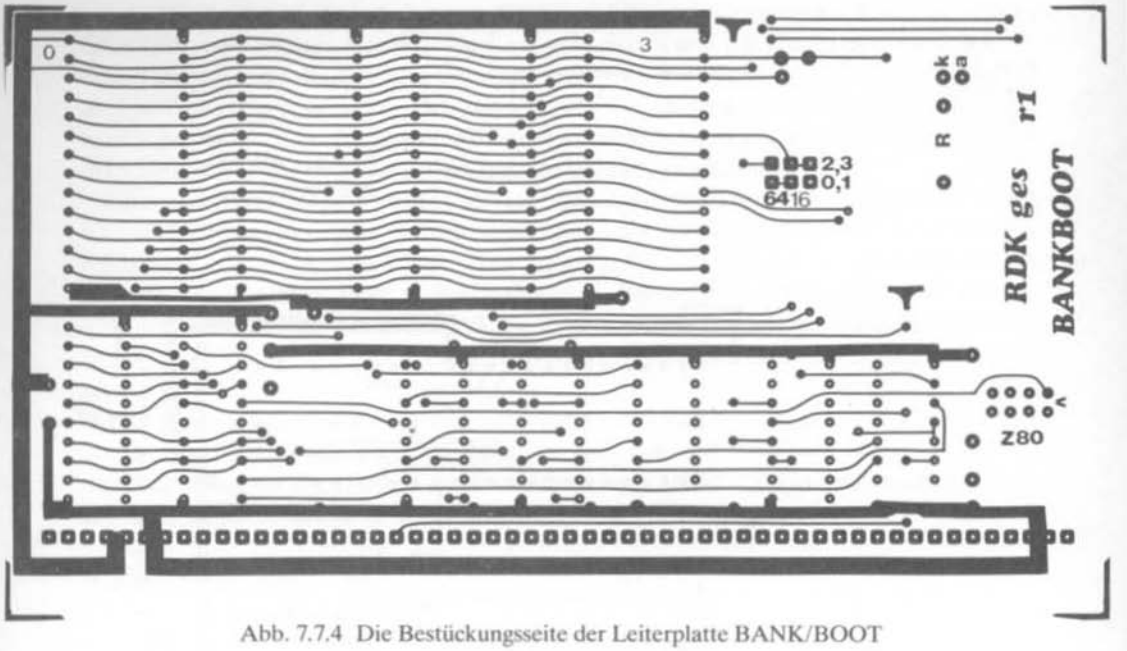


Abb. 7.7.4 Die Bestückungsseite der Leiterplatte BANK/BOOT

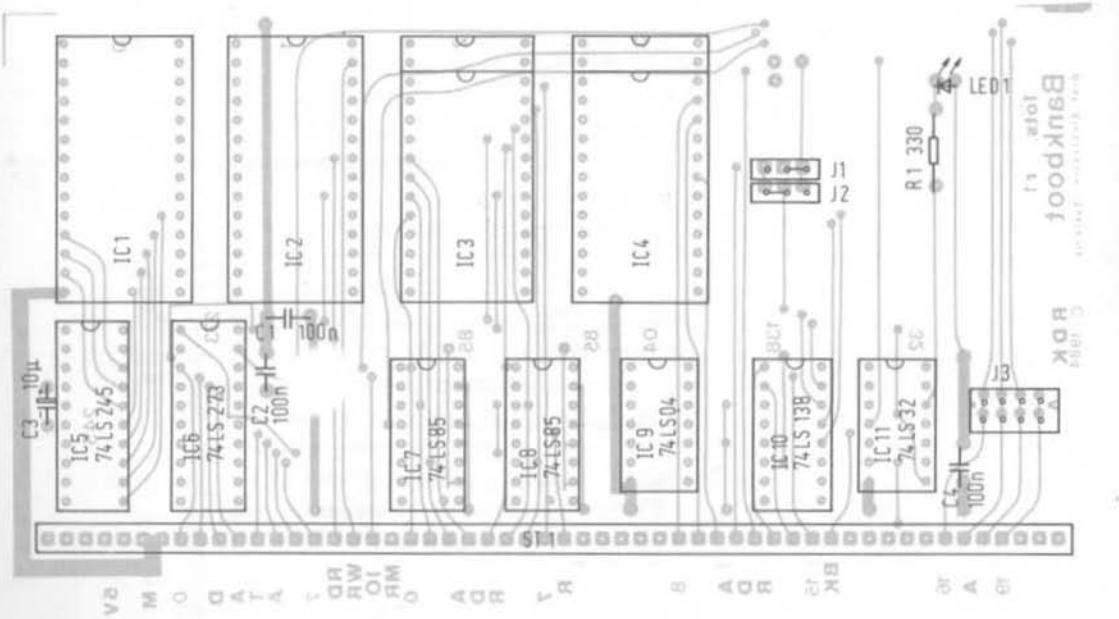


Abb. 7.7.5 Der Bestückungsplan der Baugruppe BANK/BOOT

4. Falls es nicht so arbeitet, wie beschrieben, kommt die Fehlersuche. Suchen Sie zunächst nach Lötfehlern. Ist durch eine Sichtkontrolle kein Fehler erkennbar, so müssen Sie anfangen zu messen. Auch in diesem Fall stehen ein paar Darstellungen des Timings zur Verfügung.

Abb. 7.7.6 zeigt den Vorgang unmittelbar nach dem Reset. Um die Pulse mit einem Skop zu sehen, triggern Sie dieses mit der steigenden Flanke des -RESET-Signals. Für ein periodisches Erscheinen des Signals ist es nützlich, einen Frequenzgenerator zur Verfügung zu haben. Sie müßten dann den NE555 der CPU-Baugruppe entfernen und einen 100-kHz-Takt anlegen.

In der Abbildung erkennt man, daß schon bei den ersten Zugriffen das BANKEN-Signal aktiv wird. Sie können das auch mit dem Prüfstift feststellen, es müssen Pulse nach dem Reset nachweisbar sein.

Abb. 7.7.7 zeigt einen gröberen Ausschnitt. Etwas später muß zum BOOTR-EPROM ein Schreibzugriff erfolgen, denn das Programm des EPROMs wird nun ins RAM übertragen. Beim Schreibzugriff muß das BANKEN-Signal auf 1 liegen.

Abb. 7.7.8 zeigt einen noch weniger gespreizten Auszug. Man sieht, daß nach dem Einschalten eine Reihe von -WR-Pulsen auftauchen, sie übertragen einen Teil des EPROM-Inhalts in den RAM-Bereich ab Adresse $8000. Man kann die Pulse mit dem Skop ansehen, sie erscheinen kurz nach dem Reset. Ist das Verschieben beendet, folgt das Suchen nach dem Grundprogramm-EPROM.

Hinweis:

Die BANK/BOOT-Baugruppe ist bei uns die einzige Baugruppe, die das BANKEN-Signal erzeugen darf. In späteren Baugruppen des NDR-KLEIN-Computers kann es aber von Nutzen sein, diese Einschränkung aufzuheben. Um das erfüllen zu können, muß ein Buffer mit offenem Kollektor in die Leitung BANKEN geschaltet werden, so daß nun am Bus mehrere solcher Schaltungen anliegen können. *Abb. 7.7.9* zeigt die Schaltungsergänzung. Diese Erweiterung ist z. B. für die Farb-Baugruppe COL 256 nötig, die sich auch in den Hauptspeicher einblenden darf. Wenn man die hier gezeigte BANK/BOOT-Baugruppe verwenden will, kann man sich auch einfach dadurch behelfen, daß man eine Diode (D) anstelle des ICs 7405 einbaut. Die Katode dieser Diode wird dann mit Pin 11 des IC 11 verbunden, die Anode liegt am BANKEN-Signal. Um den Strom durch diese Diode auf das verträgliche Maß zu reduzieren, muß man an allen ROA64 und RAM64/256-Baugruppen bis auf eine den Widerstand an der BANKEN-Leitung entfernen. Nur eine Baugruppe darf dort einen 4.7-kΩ-Widerstand tragen.

Diese Änderungen sind, wie bereits erwähnt, nur für die Zukunft gedacht. Unser momentanes Vorhaben, der Betrieb mit CP/M-68k, setzt diese Änderungen nicht voraus.

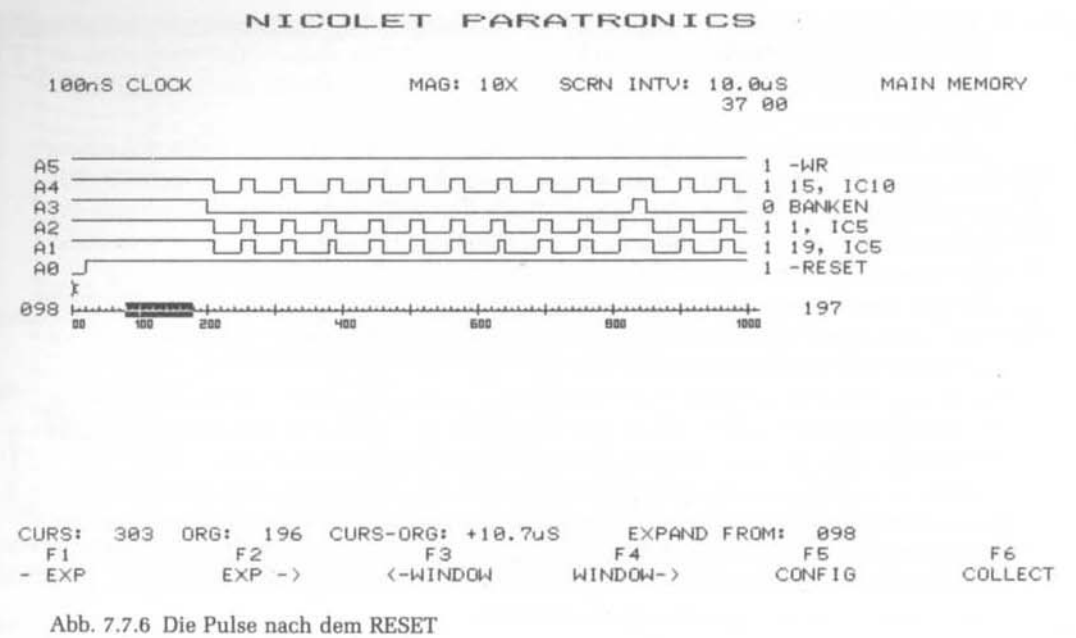


Abb. 7.7.6 Die Pulse nach dem RESET

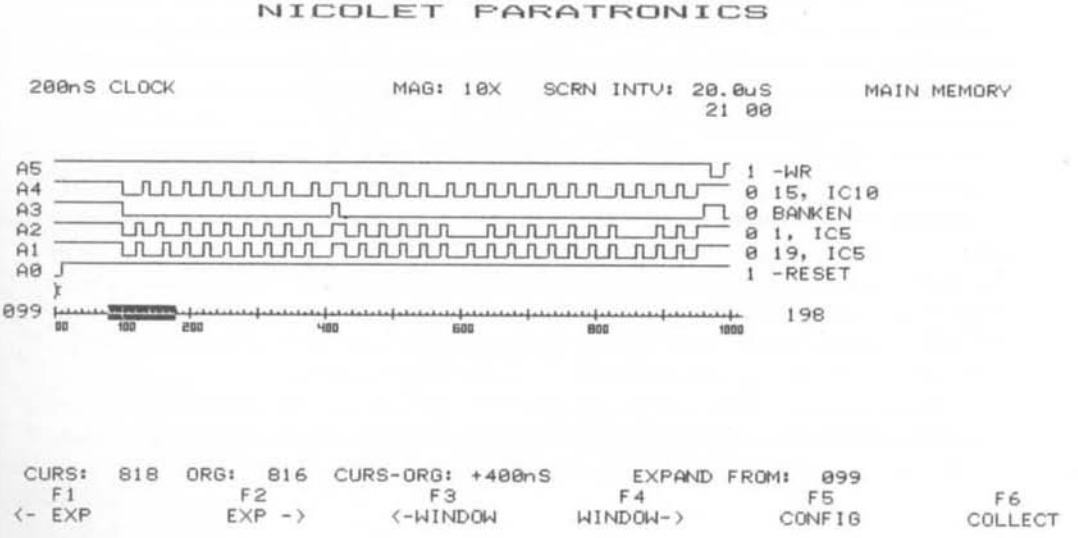


Abb. 7.7.7 Die ersten 20 µs nach denm RESET

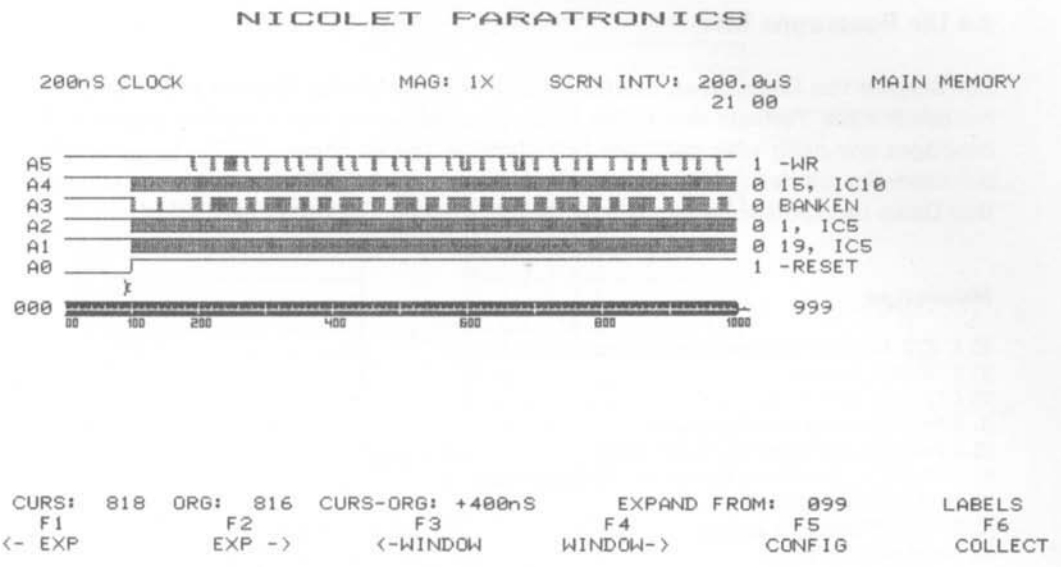


Abb. 7.7.8 Die ersten 200 µs nach dem RESET

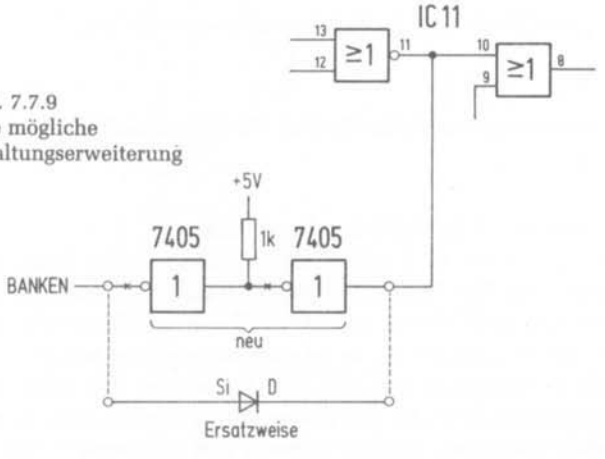


Abb. 7.7.9
Eine mögliche
Schaltungserweiterung

7.8 Die Baugruppe KEY

Zur Eingabe von Daten ist an den NDR-KLEIN-Computer eine Tastatur angeschlossen. Nachdem diese Tastatur ihre Daten nicht unmittelbar auf den Datenbus legen kann, benötigen wir dafür eine geeignete Schnittstelle. Die Baugruppe KEY ist eine solche Schnittstelle, mit der sich unterschiedliche Tastaturen anschließen lassen; sie müssen ihre Daten im ASCII abliefern, wie es bei den meisten Tastaturen der Fall ist.

Materialliste:

IC 1, IC 2 74LS86, Exklusiv-Oder-Glieder
IC 3 74LS04, Inverter
IC 4 74LS74, D-Flip-Flop
IC 5 74LS00, Nand-Verknüpfungen
IC 6 74LS245, bidirektionale Bus-Treiber
IC 7 74LS374, Zwischenspeicher mit TRI-State-Ausg.
IC 8 74LS32, Oder-Glieder
IC 9,IC 10 74LS85, Vergleicher
2x 20polige IC-Fassung
2x 16polige IC-Fassung
6x 14polige IC-Fassung
AR1 8x 3.3 kΩ Widerstandsnetzwerk
R1,R2 optional 1 kΩ, 1/8 W im Bausatz nicht enthalten.
C1,C2,C4 100 nF Kondensator
C3 10 µF Kondensator
1x DIL-Schalter, 8fach
1x Stiftleiste 36polig, gewinkelt
1x Stiftleiste 15polig, gewinkelt
1x Leiterplatte KEY
1x Tastatur mit ASCII-Anschluß: z.B. CHERRY-Low-Cost, PREH-Commander (mit NDR-Belegung) etc.
1x Verbindungskabel, Flach- oder Rund- Kabel mit Steckverbindung.

Kenndaten:

Spannung: +5 V, Stromaufnahme ohne Tastatur: 170 mA Stromaufnahme mit CHERRY-Tastatur: 280 mA

Abb. 7.8.1 zeigt das Schaltbild der Baugruppe.

Die beiden Vergleicher IC 9 und IC 10 übernehmen die Baugruppen-Auswahl. Die Adresse kann mit den Brücken JMP2 eingestellt werden. Durch das Layout ist sie bereits fest auf die Werte $FFFFFF68 und $FFFFFF69 voreingestellt. Auf der Baugruppe befinden sich zwei Eingabeports, es ist also nur der Lesebetrieb möglich. IC 8, Pin 10 führt dazu die Verknüpfung mit dem -RD Signal aus. Mit Hilfe des Signals A0 wird entweder der Baustein IC 6 oder IC 7 ausgewählt. Liegt A0 auf 0, so wird über IC 8, Pin 3 der Baustein IC 7 selektiert; liegt A0 aber auf 1, wird über IC 8, Pin 6 der Baustein IC 6 angesprochen.

Die Tastatur-Daten gelangen über eine Gruppe von Exklusiv-Oder-Verknüpfungen an die D-Eingänge des Zwischenspeichers IC 7. Die Exklusiv-Oder Glieder IC 1 und IC 2

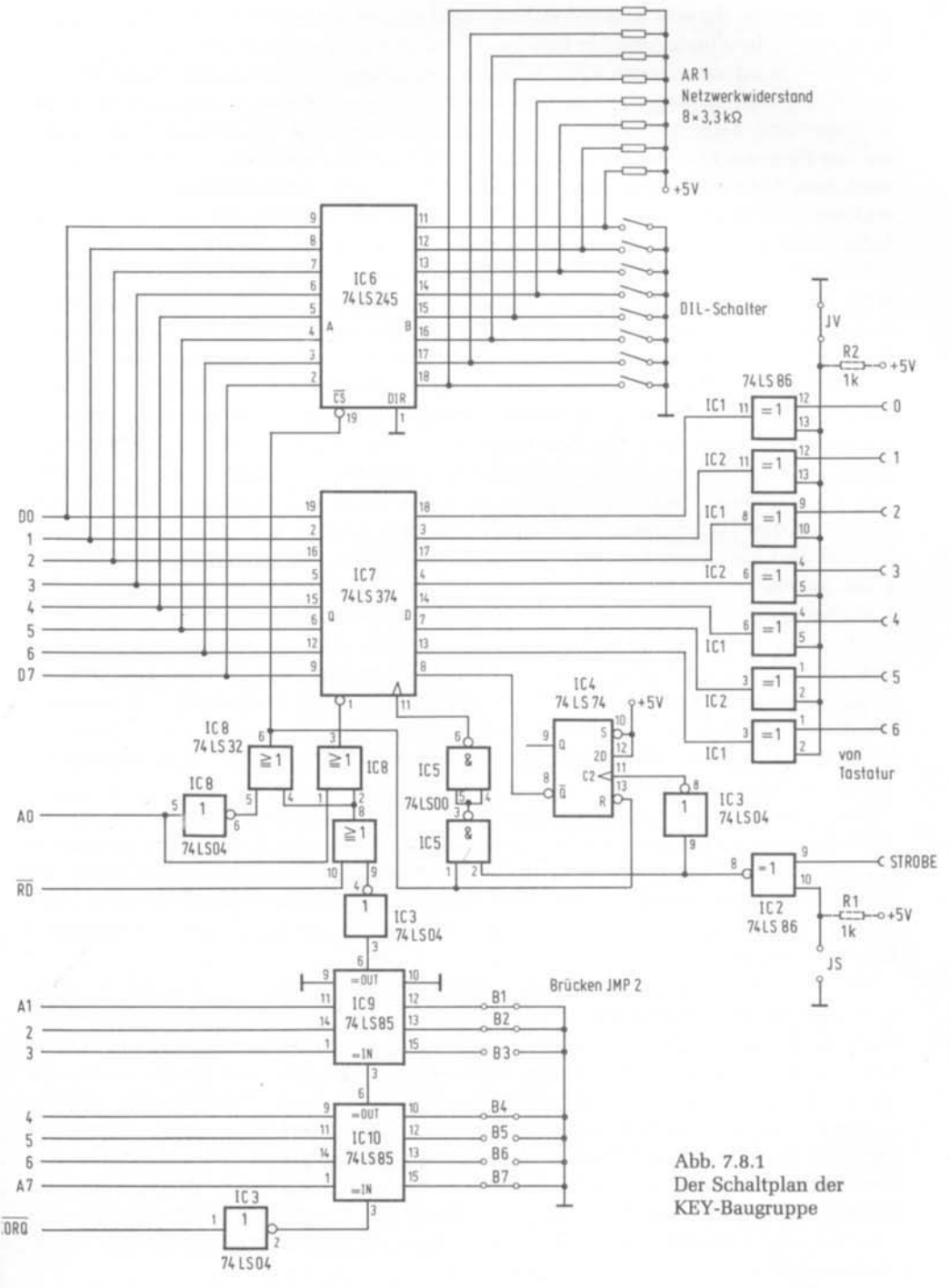


Abb. 7.8.1
Der Schaltplan der KEY-Baugruppe

ermöglichen es, die ankommenden Tastatur-Daten wahlweise zu invertieren. Abhängig ist dies von der Einstellung der Brücke „JV"; ist sie eingesetzt, werden die Daten nicht invertiert. Wird sie dagegen offen gelassen, gelangt ein 1-Signal an die Bausteine IC 1 und IC 2, damit werden alle eingehenden Signale invertiert am Ausgang der Bausteine weitergeleitet. Falls die Brücke offen bleibt, sollte man den Widerstand R2 einsetzen, der den Pegel am Eingang der Bausteine auf +5 V anhebt. Leider ist z.Z. dafür im Layout noch kein Platz vorgesehen; man kann ihn jedoch leicht in die Bohrung für die Brücke einlöten und dann mit der Leitung +5 V verbinden, die sich auf der Oberseite der Leiterplatte in der Nähe der Brücke befindet.

Die Tastatur liefert ein sogenanntes Strobe-Signal, wenn die Tastatur-Daten gültig sind. Dieses Signal dient dann als Übergabesignal zum Einspeichern der Daten in den Zwischenspeicher. *Abb. 7.8.2* zeigt ein Schema. Ein positiver Puls am Strobe-Ausgang der Tastatur führt mit seiner steigenden Flanke (1) dazu, daß der Ausgang Q-Quer des D-Flip-Flops IC 4 ein 0-Signal führt. Dabei dient die durch das IC 2 hergestellte Eingangsbeschaltung für eine Anpassung an beide Möglichkeiten für die Polarität des Strobe; ist die Brücke JS offen, wird der Puls invertiert. Wenn man eine Tastatur mit negativem Strobe-Puls verwendet, muß man die Brücke einsetzen. Am Ausgang IC 2, Pin 8 liegt also ein negativer Strobe-Puls an. Der wird durch das IC 3 nochmals invertiert und gelangt an den Takt-Eingang des D-Flip-Flops. Eine positive Flanke an diesem Eingang bewirkt die Übernahme des am D-Eingang liegenden Signals; in unserem Fall also +5 V. Somit gelangt bei der positiven, also ansteigenden Signalflanke des Strobe-Signals eine 1 an den Q-Ausgang. Der Q-Quer-Ausgang, Pin 8 erhält damit ein 0-Signal.

Das Strobesignal gelangt außerdem noch an den Pin 2 des IC 5. IC 5 ist als Und-Glied geschaltet. Dabei bilden Pin 1 und Pin 2 die Eingänge und Pin 6 den Ausgang. Wird die Baugruppe nicht über die Adresse $FFFFFF69 angesprochen, liegt an Pin 1 ein 1-Signal. Die Und-Bedingung ist also erfüllt, wenn gleichzeitig Pin 2 ein 1-Signal führt. Im Ruhezustand ist das der Fall, denn am Ausgang IC 2, Pin 8 liegt ein 1-Signal, wenn der Strobe am Eingang auf 0 liegt.

Damit liegt auch am Ausgang des Und-Gliedes ein 1-Signal, das an Pin 11, den Takteingang des Zwischenspeichers, gelangt. Kommt nun der Strobe, wird ein negtiver Puls am Eingang Pin 11, IC 7, also am Zwischenspeicher-Takt-Eingang, erscheinen. Der Zwischenspeicher übernimmt die Daten aber erst mit der steigenden Flanke am Takteingang, so daß sie erst bei der fallenden Flanke des ursprünglichen Strobe-Signals übernommen werden.

Wurden von der CPU die Daten der Adresse $FFFFFF68 gelesen, kann am Bit 7 ihre Gültigkeit erkannt werden. Besitzt es den Wert null, wurden Daten eingeschrieben. Man kann die Daten dann auswerten. Um zu verhindern, daß man bei einem Tastendruck die Daten mehrfach ausliest, muß es nun möglich sein, dieses Bit 7 nach dem Einlesen auch wieder zu löschen. Dies geschieht über den CLR-Eingang des D-Flip-Flops, Pin 13, IC 4.

Wenn man die Adresse $FFFFFF69 anspricht und dort Daten liest, gelangt ein negativer Puls an den CLR-Eingang. Der Ausgang Q-Quer geht damit wieder auf 1. Mit der steigenden Flanke des negativen Pulses (siehe „READ-Quer des Port-Schalters" im Diagramm), werden die neuen Daten wieder in den Zwischenspeicher übernommen und Bit 7 auf 1 gesetzt. Stehen die Daten nicht mehr am Tastaturausgang an, sind die restlichen Datenbits 0 bis 6 dann undefiniert.

Am Datenport $FFFFFF69 kann zusätzlich die Stellung von 8 DIL-Schaltern abgefragt werden. DIL ist die Abkürzung für Dual In Line und beschreibt die Gehäuseform; zu Deutsch „zwei (Reihen von Anschlüssen) in einer Linie".

Die Schalterstellung der DIL-Schalter spielt beim Betrieb mit dem Grundprogramm keine Rolle. Man kann sie selbst für beliebige Zwecke verwenden und abfragen. Beispiel:

```
START:
        MOVE.B $FFFFFF69,D0   * Inhalt der DIL-Schalter laden
        RTS                   * Ergebnis in D0.B
```

Abb. 7.8.3 zeigt die Lötseite der Leiterplatte und *Abb. 7.8.4* die Bestückungsseite. In *Abb. 7.8.5* ist der Bestückungsplan zu sehen.

Aufbau und Test:

1. Einlöten aller passiven Bauteile und aller IC-Fassungen. Achtung, auf der Leiterplatte befinden sich Bohrungen für drei weitere ICs, die jedoch nicht angeschlossen sind. Sie sind für Erweiterungen oder für spezielle Tastaturen gedacht. Im Normalfall bleiben sie unbestückt.
2. Spannung anlegen und auf Kurzschluß prüfen.
3. Spannung ausschalten, alle ICs einsetzen und die Tastatur noch nicht verbinden. Zum Test verwendet man die GDP64, die ROA64, sowie die CPU68K mit dem Grundprogramm. Jetzt folgt das Einschalten. Das Grundprogramm muß sich melden. Sollte dies nicht der Fall sein, liegt ein Kurzschluß zwischen zwei Leiterbahnen vor. Prüfen Sie die Leiterplatte entsprechend.
4. Die Spannung ausschalten und die Tastatur anschließen. Es gibt eine ganze Reihe verschiedener Arten von Tastaturen. Für die CHERRY-Tastatur und die PREH-Commander-Tastatur finden Sie hier Anschlußschemen. Falls Sie eine andere Tastatur verwenden, besorgen Sie sich bitte ein Anschlußschema vom Hersteller.

   Wie bereits am Anfang dieses Kapitels erwähnt, muß unsere Tastatur ihre Daten nach ASCII abliefern. *Abb. 7.8.6* zeigt die ASCII-Tabelle. Dieser Code ist nach DIN 66003 unter ISO-7-Bit-Code genormt. Allen Buchstaben, Ziffern und Zeichen werden Codes zugeordnet. Die Codes 0 bis dezimal 31 sind Steuerzeichen, die dem Rechner eine Anweisung geben und kein druckbares Zeichen darstellen.

   Abb. 7.8.7 zeigt das Verbindungsschema CHERRY-Tastatur nach KEY. Dabei liefern die Bits 1..7 die Daten nach ASCII; Bit 1 ist die niederwertigste Stelle. Die Tastatur besitzt noch einige zusätzliche Leitungen, wie AKD, etc., die jedoch offen bleiben können. Die Break-Taste besitzt eine eigene Leitung, man kann sie z.B. an den RESET-Eingang der CPU-Baugruppe anschließen, falls der Reset von der Tastatur auszulösen sein soll. Doch Achtung, allzu leicht gerät man dann aus Versehen auf diese Taste!

   Während die CHERRY-Low-Cost-Tastatur eine recht einfache Tastatur darstellt, ist die PREH-Commander-Tastatur schon komfortabler. Sie ist mit einer speziellen Belegung für den NDR-KLEIN-Computer erhältlich.

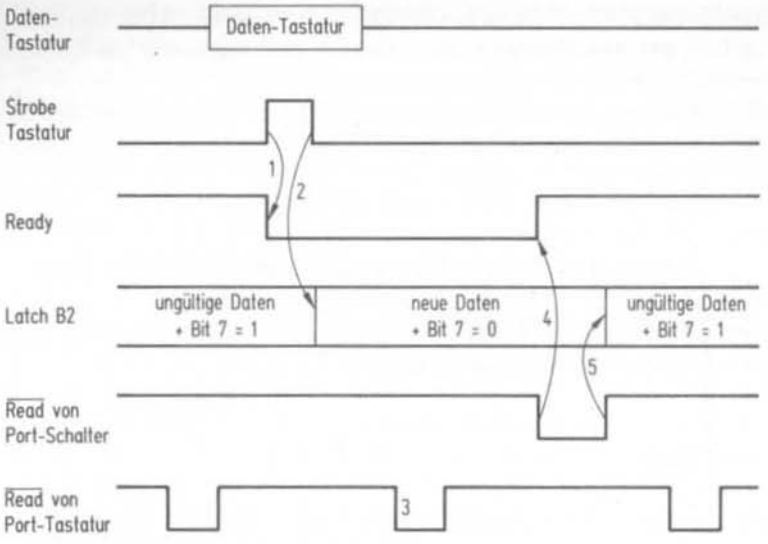


Abb. 7.8.2 Der Ablauf bei einer Tasteneingabe

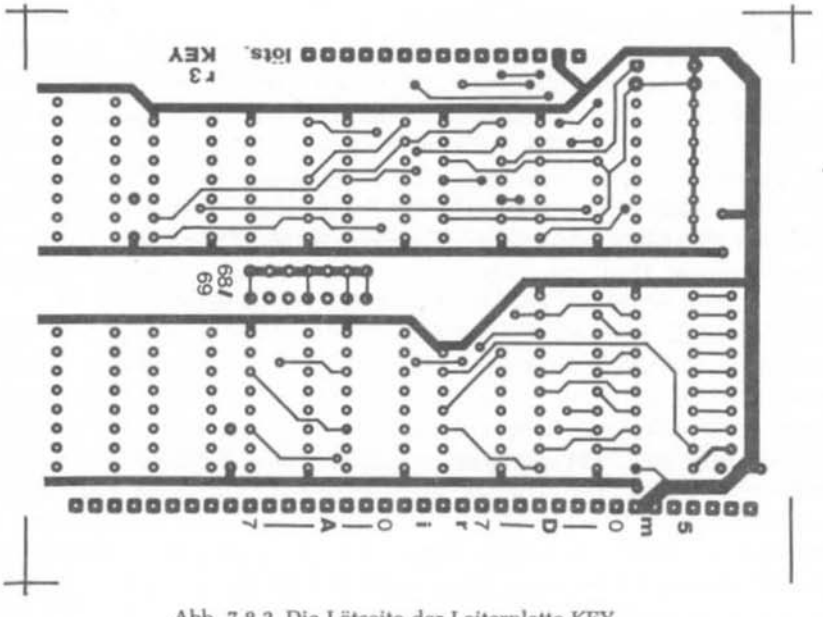


Abb. 7.8.3 Die Lötseite der Leiterplatte KEY

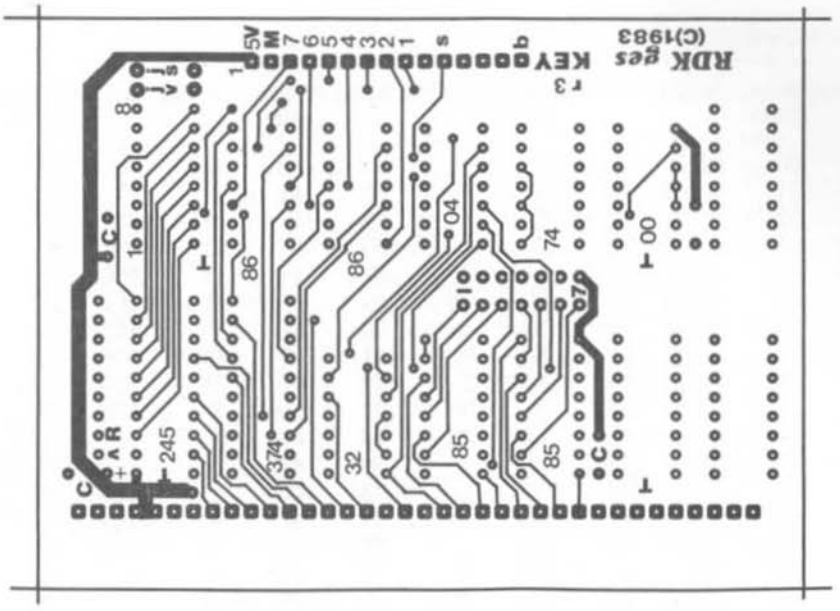


Abb. 7.8.4 Die Bestückungsseite der Leiterplatte KEY

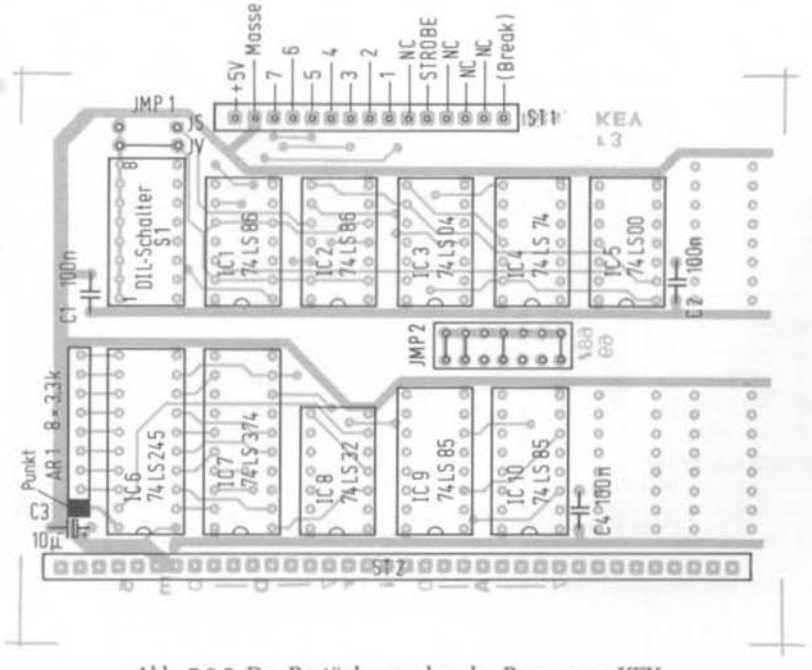


Abb. 7.8.5 Der Bestückungsplan der Baugruppe KEY

Dezimal	Sedez. (Hex)	ASCII	Dezimal	Sedez. (Hex)	ASCII	Dezimal	Sedez. (Hex)	ASCII	Dezimal	Sedez. (Hex)	ASCII
0	00	NUL	32	20		64	40	@	96	60	´
1	01	SOH	33	21	!	65	41	A	97	61	a
2	02	STX	34	22	"	66	42	B	98	62	b
3	03	ETX	35	23	#	67	43	C	99	63	c
4	04	EOT	36	24	$	68	44	D	100	64	d
5	05	ENQ	37	25	%	69	45	E	101	65	e
6	06	ACK	38	26	&	70	46	F	102	66	f
7	07	BEL	39	27	'	71	47	G	103	67	g
8	08	BS	40	28	(	72	48	H	104	68	h
9	09	HT	41	29	)	73	49	I	105	69	i
10	0A	LF	42	2A	*	74	4A	J	106	6A	j
11	0B	VT	43	2B	+	75	4B	K	107	6B	k
12	0C	FF	44	2C	,	76	4C	L	108	6C	l
13	0D	CR	45	2D	-	77	4D	M	109	6D	m
14	0E	SO	46	2E	.	78	4E	N	110	6E	n
15	0F	SI	47	2F	/	79	4F	O	111	6F	o
16	10	DLE	48	30	0	80	50	P	112	70	p
17	11	DC1	49	31	1	81	51	Q	113	71	q
18	12	DC2	50	32	2	82	52	R	114	72	r
19	13	DC3	51	33	3	83	53	S	115	73	s
20	14	DC4	52	34	4	84	54	T	116	74	t
21	15	NAK	53	35	5	85	55	U	117	75	u
22	16	SYN	54	36	6	86	56	V	118	76	v
23	17	ETB	55	37	7	87	57	W	119	77	w
24	18	CAN	56	38	8	88	58	X	120	78	x
25	19	EM	57	39	9	89	59	Y	121	79	y
26	1A	SUB	58	3A	:	90	5A	Z	122	7A	z
27	1B	ESC	59	3B	;	91	5B	[	123	7B	{
28	1C	FS	60	3C	<	92	5C	\	124	7C	|
29	1D	GS	61	3D	=	93	5D	]	125	7D	}
30	1E	RS	62	3E	>	94	5E	^	126	7E	¬
31	1F	US	63	3F	?	95	5F	_	127	7F	DEL
Deutscher Satz:											
91	5B	Ä	123	7B	ä	126	7C	ß			
92	5C	Ö	124	7C	ö						
93	5D	Ü	125	7D	ü						
Zweite Bedeutung:											
17	11	XON									
19	13	XOFF									

Abb. 7.8.6 Die ASCII-Zeichen

Brücken	KEY	CHERRY Low-Cost-Tastatur
JV einsetzen JS offenlassen	+5V 1 — 1	+5V
	M 2 — 2	Masse
	7 3 — 3	Bit 7
	6 4 — 4	Bit 6
	5 5 — 5	Bit 5
	4 6 — 6	Bit 4
	3 7 — 7	Bit 3
	2 8 — 8	Bit 2
	1 9 — 9	Bit 1
	10	offen 10	Enable I
	S 11 — 11	Strobe
	12	offen 12	AKD
	13	offen 13	Enable II
	14	14
	b 15 — 15	Break

Abb. 7.8.7
Der Anschluß der CHERRY-Low-Cost-Tastatur

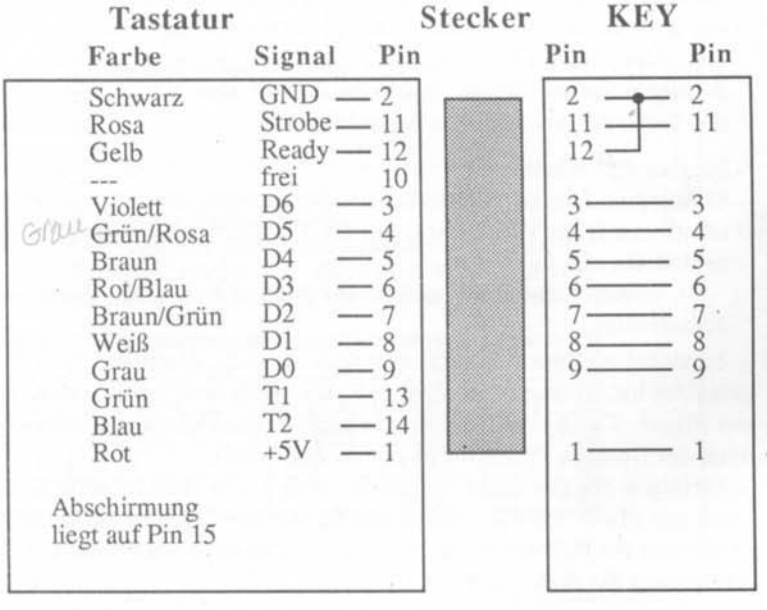


Abb. 7.8.8
Der Anschluß der Preh-Commander-Tastatur

Abb. 7.8.8 zeigt das Anschlußschema und *Abb.7.8.9* eine wichtige Leitung, die man zum Arbeiten einlöten muß.

Die PREH-Tastatur besitzt einen 15poligen Cannon-Verbinder (Buchse), ein Gegenstück kann man im Fachhandel erhalten.

Zur Modifikation wird entweder die Buchse aufgeschraubt oder die Leitung am Stiftverbinder durchgeführt.

Die PREH-Commander-Tastatur verfügt über Funktionstasten, die schon mit Codes für den NDR-KLEIN-Computer belegt sind.

Abb. 7.8.10 zeigt eine Übersicht, *Abb. 7.8.11* die genaue Zeichenbelegung der einzelnen Tasten und *Abb. 7.8.12* die ASCII-Belegung. Dabei werden vier Tastenmodi unterschieden, die in Abhängigkeit von dem Zustand der Tasten SHIFT und CTRL, sowie des Schalters ALPHA-LOCK Gültigkeit erlangen. Der Control-Mode ergibt sich, wenn man die CTRL-Taste zusätzlich drückt; der Shift-Mode, wenn man die SHIFT-Taste zusätzlich drückt; der Normal-Mode, wenn keine der beiden Tasten gedrückt ist. Wird der ALPHA-LOCK-Schalter betätigt, rastet er ein, und es ergibt sich ein weiterer Mode, der sich vom Shift-Mode dadurch unterscheidet, daß hier nur Buchstaben in Großbuchstaben gewandelt werden. Will man die Ziffern in Sonderzeichen verwandeln, muß man bei Alpha-Lock zusätzlich die Shift-Taste drücken. Manche Funktionstasten liefern zwei Codes (für die Editor-Funktionen im Grundmenue) nacheinander.

Abb. 7.8.13 zeigt die Tastenbelegung des Ziffernfeldes und *Abb. 7.8.14* den jeweiligen Code dazu.

Prüfen Sie vor dem Einschalten nochmals sorgfältig, ob +5 V und Masse richtig angeschlossen sind, sonst kann die Tastatur zerstört werden.

5. Nun folgt der Abschlußtest. Schalten Sie den Computer ein. Es muß ein blinkendes Cursorfeld erscheinen. Wenn Sie nun z.B. die Taste „A" drüchen, muß ein kleines „a" auf dem Bildschirm erscheinen. Drücken Sie eine weitere Taste, muß es wieder verschwinden. So kann man alle Tasten einmal grob prüfen.
   Betätigen Sie bei diesem Test keine Steuertasten, wie z.B. RETURN. Jetzt können Sie die Versuche des Kapitels 2 durchführen.

6. Fehlermöglichkeiten.
   Es kommen falsche Buchstaben an. Sie drücken ein „A", und es erscheint „c". Dann ist etwas mit der Verbindung von der Tastatur zur KEY-Baugruppe nicht in Ordnung; prüfen Sie alle Leitungen.

   Es kommt überhaupt nichts an. *Abb. 7.8.15* zeigt Ihnen einen vollständigen Signalverlauf.

Zunächst können Sie an Pin 1, IC 7 messen. Nachdem sich das Grundprogramm gemeldet hat, müssen dort ständig Pulse erscheinen. Ist dies nicht der Fall, messen Sie das Signal -IORQ am Bus; es muß ebenfalls aus Pulsen bestehen (wenn auch mehr, da auch der Graphik-Prozessor angesprochen wird).

Verfolgen Sie das Signal weiter; es muß hinter dem Inverter IC 3, Pin 2 ankommen, dann am Pin 6 des IC 10 und an Pin 6 IC 9. Wichtig bei diesem Test ist nur, daß überhaupt ein Puls ankommt, es genügt daher auch ein Prüfstift. Verwenden Sie für die Verfolgung die Abb. 7.8.1.

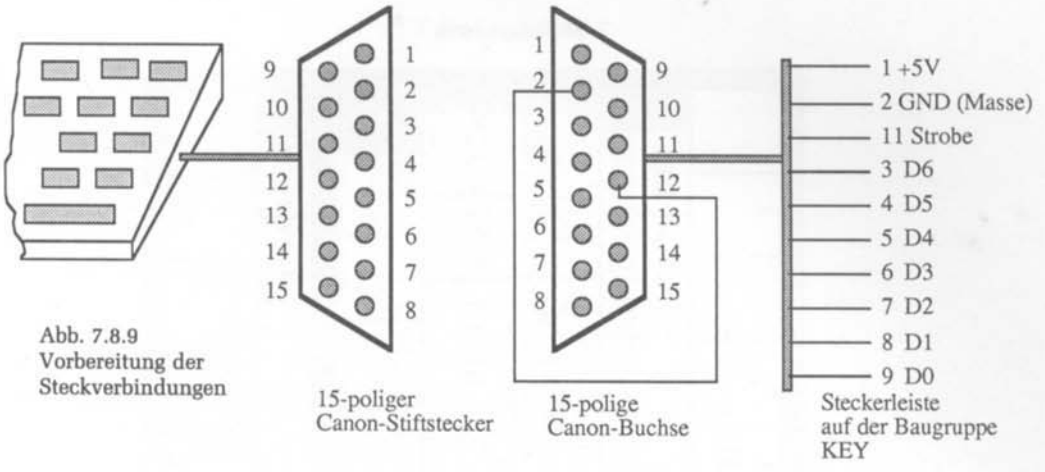


Abb. 7.8.9
Vorbereitung der Steckverbindungen

/ bedeutet CONTROL
*** bedeutet, daß sich die Taste auf den Texteditor des 68008-Systems (Wordstar-Kompatibel) bezieht.**

Taste	Normal		Mit SHIFT betätigt	
1	W cr für die Menüs		W cr	
2 *	Hilfe	/J	Hilfe	/J
3 *	Ende Texteditor	/KX	Ende Texteditor	/KX
4 *	Zeichensatz umschalten	/P	Zeichensatz umschalten	/P
5 *	Einfügemodus	/V	Einfügemodus	/V
6 *	Suchen nach String	/QF	Weitersuchen	/L
7 *	Suchen und Ersetzen	/QA	Weitersuchen	/L
8 *	frei		frei	
9 *	frei		frei	
S1 *	Ein Zeichen links löschen (= DEL-Taste)		Ab Cursor rechts alles löschen	/T
S2 *	Seite zurück	/R	Seite zurück	/R
S3 *	Seite vor	/C	Seite vor	/C

Alle Cursortasten sind im Editor belegt.
Mit LOCK wird die Tastatur auf Großbuchstaben umgestellt, mit CTRL/SHIFT/LOCK wieder auf Kleinbuchstaben zurück.
Im Programmteil "Ändern" bewirken die Cursortasten ein CR

Abb. 7.8.10 Die Belegung der Tasten

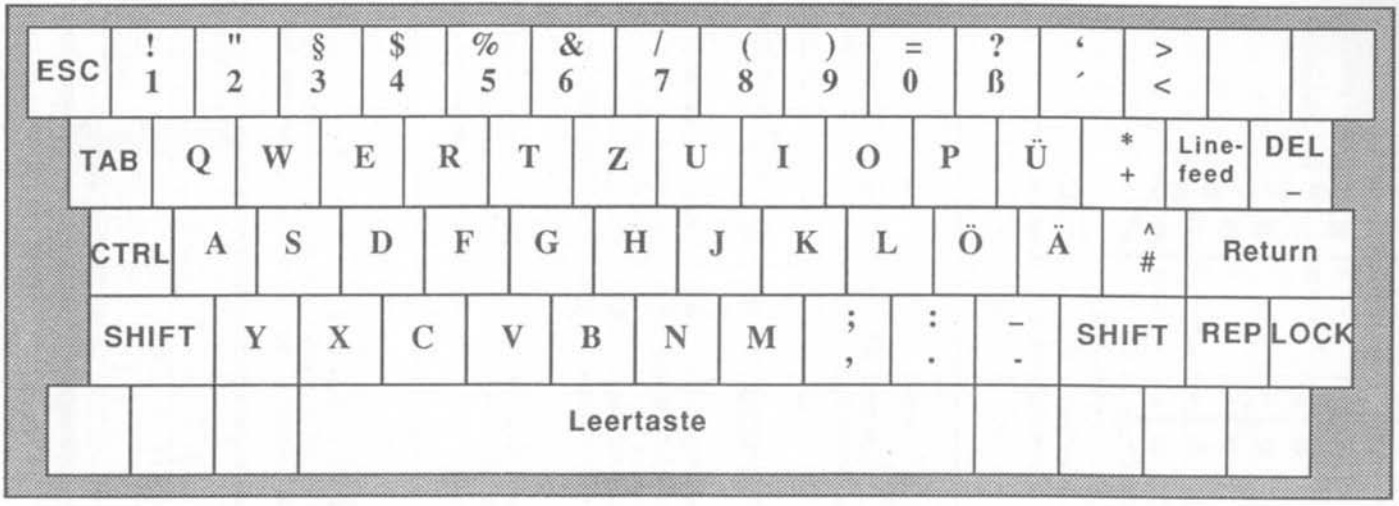


Abb. 7.8.11 Die Belegung des Tastenfeldes

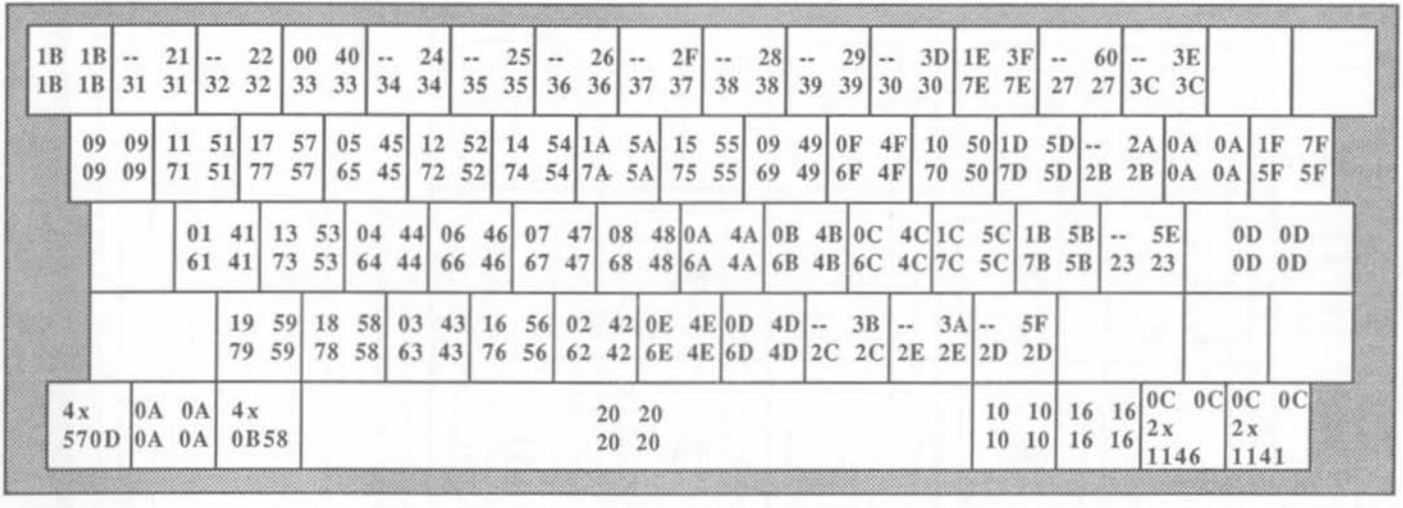


Abb. 7.8.12 Die Codes der Tasten

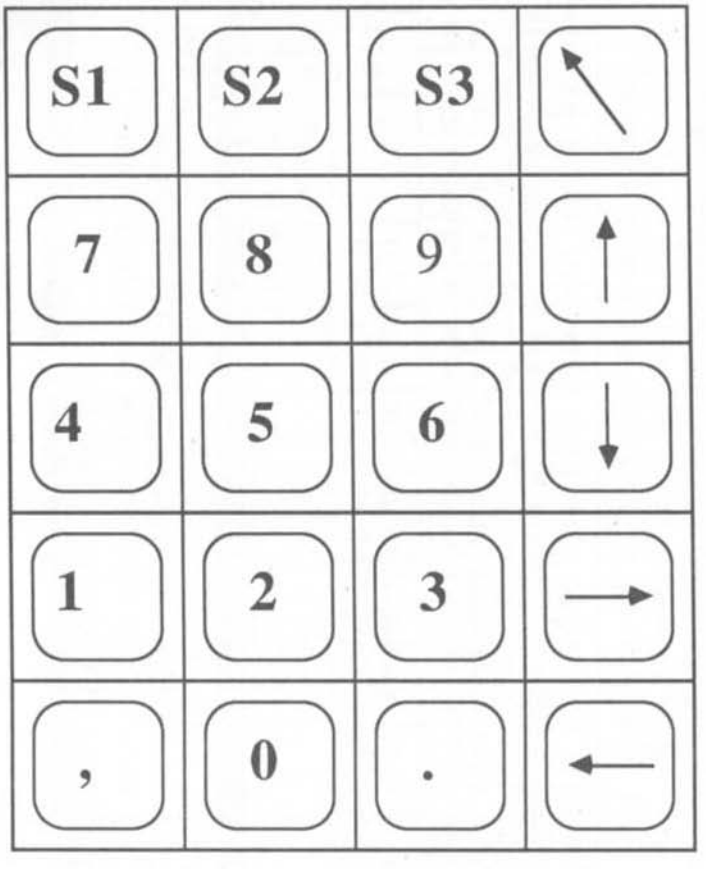


Abb. 7.8.13 Die Belegung des kleinen Tastenfeldes

14 14 7F 7F	-- 12 12 12	-- 03 03 03	-- 11 43 11 52 11 52
37 37 37 37	38 38 38 38	39 39 39 39	19 1A 05 05
34 34 34 34	35 35 35 35	36 36 36 36	0E 17 18 18
31 31 31 31	32 32 32 32	33 33 33 33	07 06 04 04
2C 2C 2C 2C	30 30 30 30	2E 2E 2E 2E	7F 01 13 13

Abb. 7.8.14
Die Codes
auf dem kleinen
Tastenfeld

xx xx xx xx	**Code bei Drücken der CTRL-Taste** **Code bei Drücken der SHIFT-Taste** **Code ohne Zusatztaste** **Code wenn Alphalock gedrückt wurde**

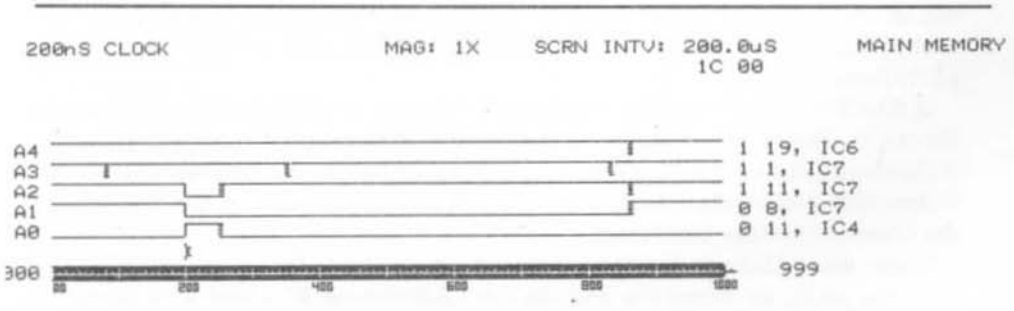


Abb. 7.8.15 Der Ablauf bei einer Tasteneingabe, wenn das Grundprogramm arbeitet

Wenn man eine Taste drückt, so muß ein Strobe am Eingang der KEY-Baugruppe erscheinen; prüfen Sie das. Dieser Strobe muß auch an Pin 11, IC 4 und an Pin 11, IC 7 ankommen.

Messen Sie den Eingang Pin 8, IC 7. Normalerweise liegt dort ein 1-Signal, da die Daten sofort von der CPU abgeholt werden, wenn das Grundprogramm aktiv ist.

Wenn man eine Taste drückt, muß ein 0-Signal für ca. 130 µs erscheinen. Bleibt es aus, so funktioniert vielleicht das Löschen von IC 4 über Pin 13 nicht. Messen Sie dort.

Wenn Sie die Möglichkeit haben, an einen bereits funktionierenden NDR-KLEIN-Computer zu kommen, ist der Test der KEY-Baugruppe einfacher. Sie setzen dann z. B. die Brücke B4 an JMP2 ein und können Ihre Karte gleichzeitig mit der des anderen Computers auf dem Bus haben. Damit können Sie jetzt ein Testprogramm schreiben.

Die Baugruppe wird bei eigesetzter Brücke B6 auf den Adressen $FFFFFF28 und $FFFFFF29 adressiert (B7, B4, B2 und B1 sind schon im Layout vorhanden).

Testprogramm:

```
START:
  MOVE.B $FFFFFF29,D0
  BRA.S START
```

Übersetzen und starten Sie dieses Programm. An Pin 19, IC 6 müssen Pulse erscheinen. Sie können die Funktion auch über das Menue IO-LESEN prüfen, indem Sie dort $FFFFFF29 als Adresse eingeben und „D" drücken. Wenn Sie dann die DIL-Schalter-Positionen verändern, so muß die eingestellte Schalterkombination im Bild anhand der dualen Darstellung ablesbar sein.

Prüfen Sie, ob der Puls am IC 4, Pin 11 ankommt.

Vergessen Sie nach der Fehlersuche nicht, die Brücke B6 wieder aufzutrennen!

7. Störungen: Wenn Sie eine Taste drücken, und es erscheinen zwei (oder mehr) Zeichen auf dem Bildschirm, liegt eine Störung vor.

   a) Der Strobe Ihrer Tastatur hat ein anderes Zeitverhalten. Kontrollieren Sie die Pulsformen. Die Daten müssen vor und nach dem Puls anliegen, der Puls muß an Pin 11, IC 4 als positiver Puls vorliegen, sonst ggf. die Brücke JS einsetzen.

   b) Die Tastatur prellt, was bei heutigen Tastaturen jedoch sehr selten ist. Prüfen Sie, ob das „Prellen" von der Stärke des Anschlages abhängt, also ob die Buchstabenzahl vom Druck und der Art des Anschlags abhängt. Ggf. muß man die Tastaturleiterplatte besser befestigen.

   c) Eine Störung gelangt über eine lange Leitung an die KEY-Baugruppe. Wenn die Brücke JS offen ist, löten Sie einen zusätzlichen Widerstand R1 an die Brücke. Den Widerstand kann man auf der Bestückungsseite der Baugruppe auflöten. Für den +5-V-Anschluß gibt es kein Lötauge, man muß ihn direkt auf die Leiterbahn, die auch auf der Oberseite entlang läuft, löten.

   Durch diese Maßnahme wird die Schaltung unempfindlicher gegen Störungen. Hilft das nicht, so setzen Sie anstelle der LS-Bausteine IC 1 und IC 2 Standard-Bausteine 7486 ein. Sie reagieren ebenfalls unempfindlicher auf Störungen.

7.9 Die Baugruppe GDP64

Die GDP64-Baugruppe stellt auch eine wesentliche Baugruppe des NDR-KLEIN-Computers dar. Sie hat die Aufgabe, die Bildschirmsteuerung zu übernehmen. Die Buchstaben G D P stehen dabei als Abkürzung für graphik display processor; also ein Prozessor, der Graphiken anzeigen kann. Prozessor deshalb, weil der Baustein auf der GDP im Prinzip ein kleiner Mikroprozessor ist, wenn auch mit einem sehr speziellen Befehlssatz.

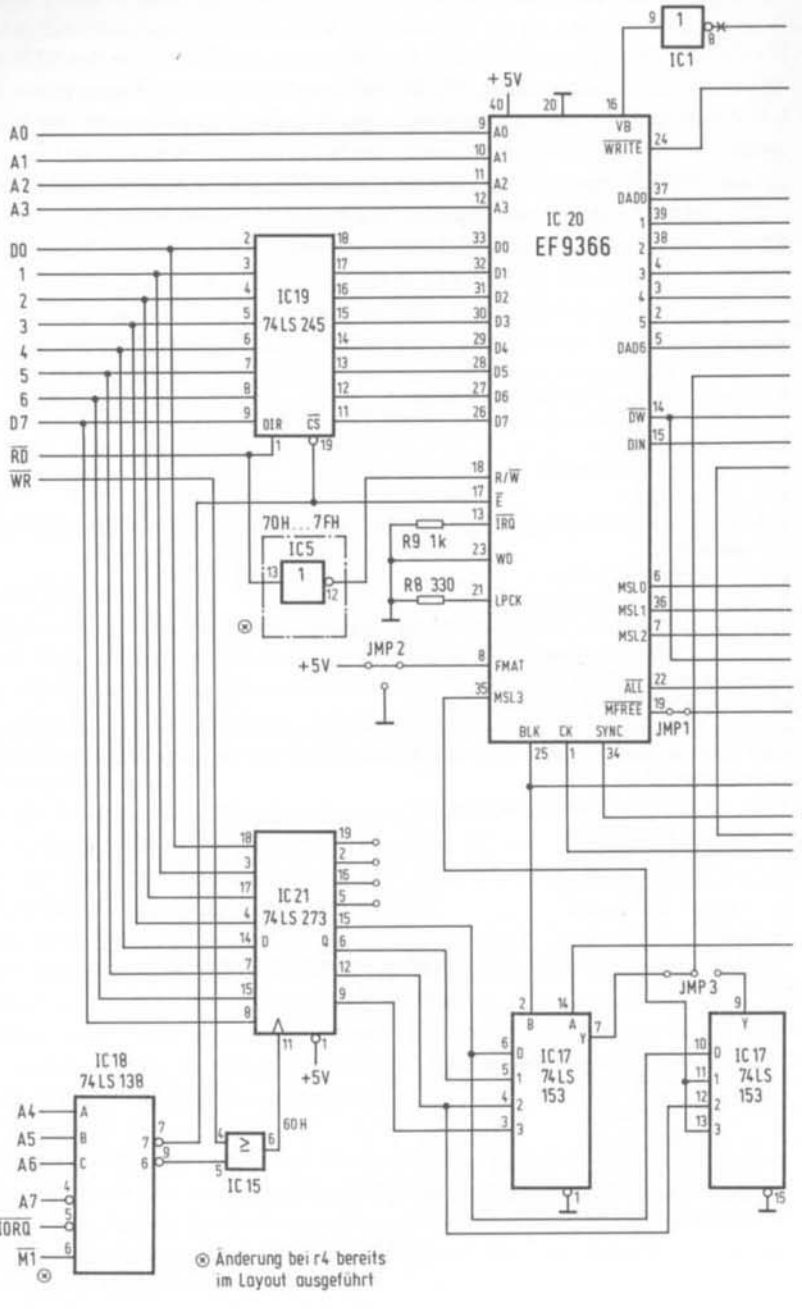
Die Zahl 64 kommt von der Größe des Bildwiederholspeichers, der ist nämlich 64 KByte groß. Damit lassen sich 4 Bildseiten mit je 512 mal 256 Bildpunkten speichern.

Materialliste:

IC 1 7405 Inverter mit offenem Kollektor
IC 2 74LS166 Schieberegister
IC 3 74LS163 Zähler
IC 4 74LS00 Nand-Glieder
IC 5 7404 Inverter
IC 6..IC 13 4164, 200 ns dynamische 64 K x 1 Speicher, Achtung, nur solche mit 128 Refresh-Zyklen verwenden!
IC 14 74LS74 D-Flip-Flop
IC 15 74LS32 Oder-Verknüpfungen
IC 16 25LS2538 1 aus 8 Dekoder mit Pol-Eingang, von AMD (Advanced Mikro Devices)
IC 17 74LS153 4 zu 1 Multiplexer
IC 18 74LS138 1 aus 8 Dekoder
IC 19 74LS245 bidirektionale Bustreiber
IC 20 EF9366 graphic display processor Achtung, auch EF 9365 oder EF 9367 einsetzbar, sie werden aber vom Grundprogramm nicht direkt unterstützt, und es müssen die Brücken JMP3 und JMP1 eingestellt werden.
IC 21 74LS273 8fach Zwischenspeicher
1x 40polige IC-Fassung
3x 20polige IC-Fassung
12x 16polige IC-Fassung
5x 14polige IC-Fassung
R1 75 Ω ca. 1/4 Watt
R2,R4,R6,R9 1 kΩ 1/8 Watt oder 1/4 Watt, wie alle weiteren
R3,R7 470 Ω
R5 150 Ω
R8 330 Ω
C1,C7 10 µF, Tantal oder kleiner Elko.
C2,C3,C4,C5,C6 100 nF
T1 BC 107, Transistor
Q1 14.000 MHz Quarz
1x Stiftleiste, gewinkelt, 36polig
1x Stiftleiste, gewinkelt, 7polig (für BAS und spez. Signale).
1x Leiterplatte GDP64

Kenndaten:

Spannungsversorgung: +5 V, Stromaufnahme: 340 mA


IC1
+5V
IC 20
EF9366
IC19
74LS245
IC21
74LS273
IC18
74LS138
IC17
74LS
153
IC17
74LS
153
IC 15
IC5
70H ... 7FH
R9 1k
R8 330
JMP 2
JMP1
JMP3
+5V
60H
A0
A1
A2
A3
D0
1
2
3
4
5
6
D7
RD
WR
A4
A5
A6
A7
IORQ
M1
DIR
CS
R/W
E
IRQ
WD
LPCK
FMAT
MSL3
VB
WRITE
DAD0
DAD6
DW
DIN
MSL0
MSL1
MSL2
ALI
MFREE
BLK
CK
SYNC
⊗ Änderung bei r4 bereits
im Layout ausgeführt

Zu Abb. 7.9.1

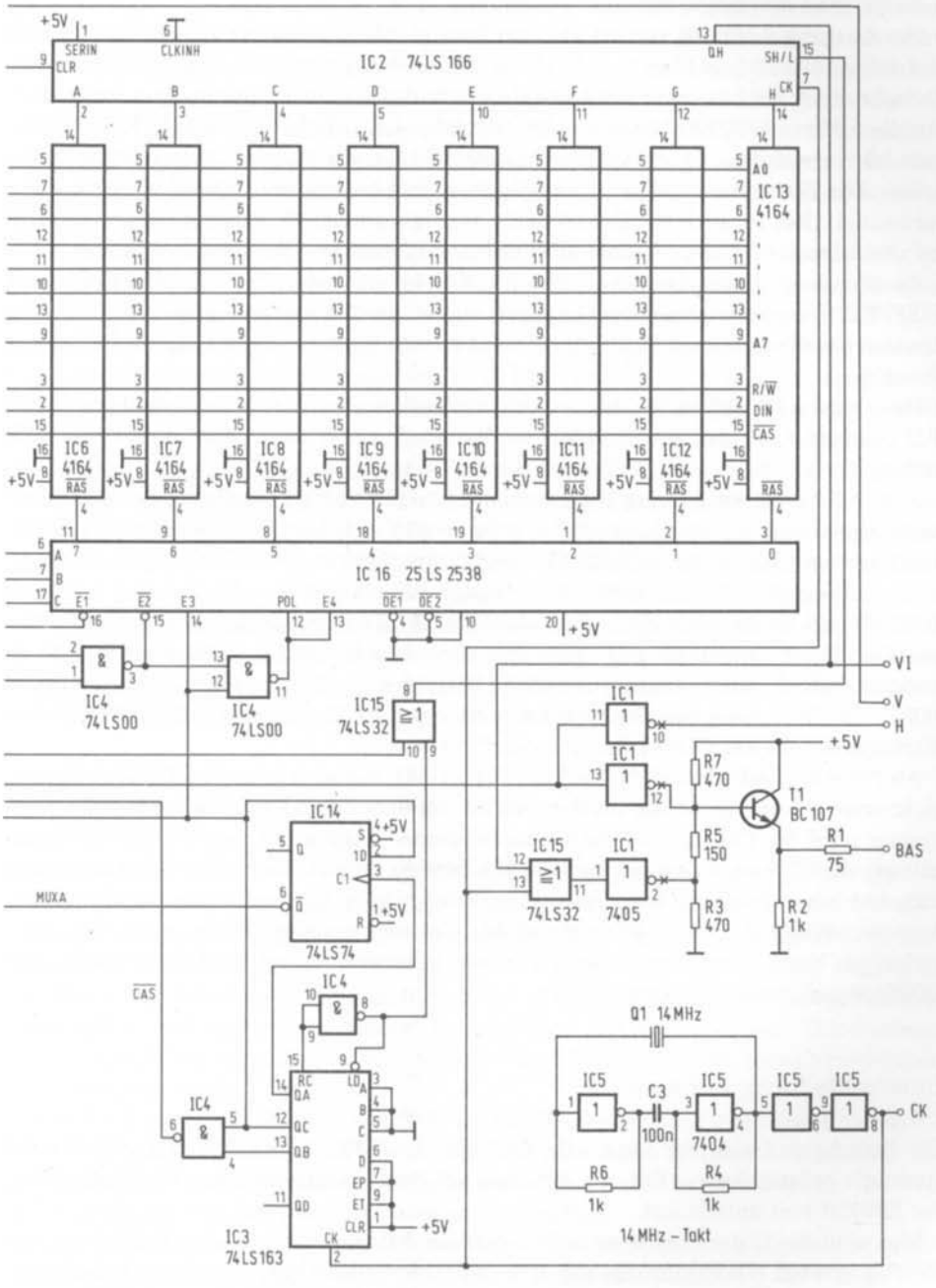


Abb. 7.9.1 Der Schaltplan der Baugruppe GDP64

Abb. 7.9.1 zeigt die komplette Schaltung der Baugruppe. Die Dekodierung der Port-Adresse wird durch den Baustein 74LS138 (IC 18) vorgenommen.

Der Ausgang 6 (Pin 9) spricht auf den Bereich $FFFFFF60 bis $FFFFFF6F an und gibt seine Information über das IC 15 an den Zwischenspeicher IC 21 weiter. Dieser Zwischenspeicher bestimmt die Auswahl der Bildseite; dabei wird im Grundprogramm nur die Adresse $FFFFFF60 verwendet. Die Adressen $FFFFFF61 bis $FFFFFF6F sollte man nie verwenden, da sie später vielleicht einmal für andere Dinge reserviert sein sollen. Die Adresse ist dabei nur als Schreib-Port belegt; beim Lesen wird sie nicht verwendet. Dies ist auch der Grund dafür, warum es möglich war, die Baugruppe KEY auf die Adressen $FFFFFF68 bis $FFFFFF69 zu legen; von ihr wird nur gelesen.

An Ausgang 7 des Dekoders (Pin 7, IC 18) wird der Bereich $FFFFFF70 bis $FFFFFF7F dekodiert. Das Signal gelangt einmal an den Freigabeeingang des bidirektionalen Datenbustreibers IC 19, Pin 19 und an den Eingang -E des Graphik-Prozessors (Pin 17).

Der Eingang R/-W (Pin 18) des Graphik-Prozessors wird durch das invertierte Signal -RD beschaltet. In der Revision r3 der Baugruppe war das Signal direkt mit -WR verbunden.

Abb. 7.9.2 zeigt ein Timing-Diagramm der jetzigen Version. Dadurch, daß das invertierte Signal -RD als R/-W verwendet wird, ergibt sich beim Schreiben ein exakterer Signalverlauf. Das Signal „SCHREIBE" liegt nun schon vor dem -IORQ-Signal an Pin 18 an. Zum Vergleich ist im Diagramm das Signal -WR eingetragen, das aber erst nach dem -IORQ-Signal erscheint. Im Datenblatt des EF9366 ist das als nicht zugelassen ausgewiesen. Der Baustein verhält sich zwar mit der alten Schaltung auch so, wie man es erwartet, jedoch benötigt man zusätzliche Wartezyklen, dß der -WR-Puls kürzer ist als -IORQ. In der neuen Schaltung kann man bei 8 MHz CPU-Takt auch völlig ohne Wartezyklen mit der GDP64-Baugruppe arbeiten.

An Pin 6 des Dekoders IC 18 liegt ein Signal -M1, das aber beim 68000/68008-System nicht erzeugt wird. Es ist nur für den Betrieb mit dem Z80 notwendig. Im 68000/68008-System wird der GDP immer dann zum Schreiben ausgewählt, wenn kein -RD-Signal anliegt, -IORQ vorhanden ist und die IO-Adresse anliegt. Beim Z80 kann dies auch während einer Interrupt-Behandlung auftreten. Dann liegt auf dem Adreßbus eine Adresse, -IORQ liegt an, -RD aber nicht. Um das auszuschließen, wird -M1 dazu verknüpft, denn immer wenn beim Z80 ein Interrupt erzeugt wird, liegt neben dem -IORQ-Signal auch das Signal -M1 an.

Unterschiedliche Formate:

Die Baugruppe ist in der Lage, alle drei ICs, den EF9365, EF9366 und den EF9367 alternativ aufzunehmen. Für das Arbeiten mit dem Grundprogramm wird jedoch nur der EF9366 voll unterstützt.

Was sind die Unterschiede zwischen den Bausteinen? Sie liegen im Bildformat. Der EF9366 erzeugt ein Bildformat mit 512 mal 256 Bildpunkten. Damit ist es bei sonst bescheidenen Ansprüchen an den Bildschirm möglich, 80 Zeichen pro Zeile unterzubringen.

Der Baustein EF9365 erzeugt ein Bildfenster von 512 mal 512 Bildpunkten. Ein normaler Fernseher, oder auch ein Monitor, arbeitet mit 625 Zeilen. Diese 625 Zeilen werden aber nicht in einem, sondern in zwei Durchläufen auf den Bildschirm gebracht. Jeder der Durchläufe dauert 20 ms. Beim ersten Durchlauf werden z. B. alle ungeraden, beim zweiten Durchlauf alle geraden Zeilennummern geschrieben. *Abb. 7.9.3* zeigt ein Schema. Man nennt das Zeilensprungverfahren. Der EF9365 erzeugt die 512 dargestellten Zeilen nach diesem Zeilensprungverfahren. Dadurch ergibt sich aber ein leichter Flimmereffekt. Der EF9366 kann wegen der kleineren Zeilenzahl auf das Zeilensprungverfahren verzichten.

Das Zeilensprungverfahren erfordert also einen Monitor mit lang nachleuchtendem Bildschirm, um den Flimmereffekt zu verhindern. Ein lang nachleuchtender Monitor läßt aber keine schnell bewegte Graphik zu; alle bewegten Objekte würden dann Fahnen hinterlassen. Daher eignet sich nur der EF9366 für die meisten unserer Experimente.

Der EF9367 ist eine Mischung aus beiden. Er kann sowohl 512 mal 256 Bildpunkte als auch 512 mal 512 Bildpunkte darstellen und per Brücke auf beide Formate eingestellt werden.

Allerdings hat die Sache auch einen Haken. Der EF9367 verwendet eine niedrigere Quarzfrequenz, nämlich 12 MHz, um ein rechteckiges Bildfenster zu erzeugen. Die Bildfenster der beiden anderen GDPs sind nämlich quadratisch. In quadratischen Bildfenstern lassen sich aber Kreise und Figuren leichter darstellen, als in rechteckigen Fenstern. Ein rechteckiges Fenster wiederum nützt natürlich den Bildschirm besser aus.

Wer dennoch einmal einen der anderen Prozessoren ausprobieren will, der findet in den folgenden Abbildungen entsprechende Hinweise: *Abb. 7.9.4* zeigt die Umstellung der Brücken für den Betrieb mit dem EF9365 und *Abb. 7.9.5* die auf den Betrieb mit dem EF9367; beide Betriebsarten (512 x 256 und 512 x 512) sind angegeben.

Speicherorganisation:

Zur Darstellung von 512 mal 256 Bildpunkten benötigt man 16 KByte Speicher (prüfen Sie den Wert nach, indem Sie 512 x 256 rechnen, und das Ergebnis durch 8 teilen).

Da auf der Baugruppe aber 64 KByte vorhanden sind, lassen sich beim EF9365 vier, beim EF9365 sowie beim EF9367 (in der Betriebsart 512 x 512) zwei Bildseiten speichern. Die Schaltung ist so konstruiert, daß es möglich ist, eine beliebige Bildseite darzustellen und auf einer anderen zu schreiben. Damit kann man flimmerfreie, bewegte Graphiken erstellen. Für die Auswahl der Bildseiten ist der Zwischenspeicher, IC 21, verantwortlich.

Abb. 7.9.6 zeigt das Schema. Die Bits 0 bis 3 sind unbelegt, und 7 bis 4 bestimmen die Bildseite. Dabei bestimmen die Werte von Bit 6 und 7 die aktuelle Schreibseite; die Werte der Bits 5 und 4 wählen die aktuelle Leseseite aus (bei 512 x 512 ist nur Bit 6 für die Schreibseite und Bit 4 für die Leseseite zuständig).

Die Umschaltung zwischen Schreib- und Leseseite wird durch das Signal BLK (siehe auch Datenblatt der Herstellerfirma THOMSON) bestimmt. Wenn BLK auf 0 liegt, wird gerade das Bild angezeigt, also aus dem Speicher gelesen. Der GDP schreibt aber nur außerhalb der Anzeige-Dauer. In *Abb. 7.9.7* finden Sie eine graphische Gegenüberstel-

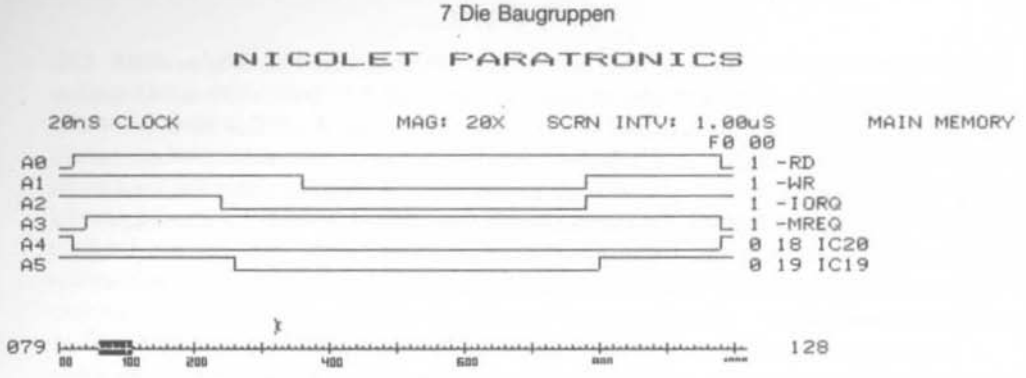


Abb. 7.9.2 Das Timing beim Schreibzugriff

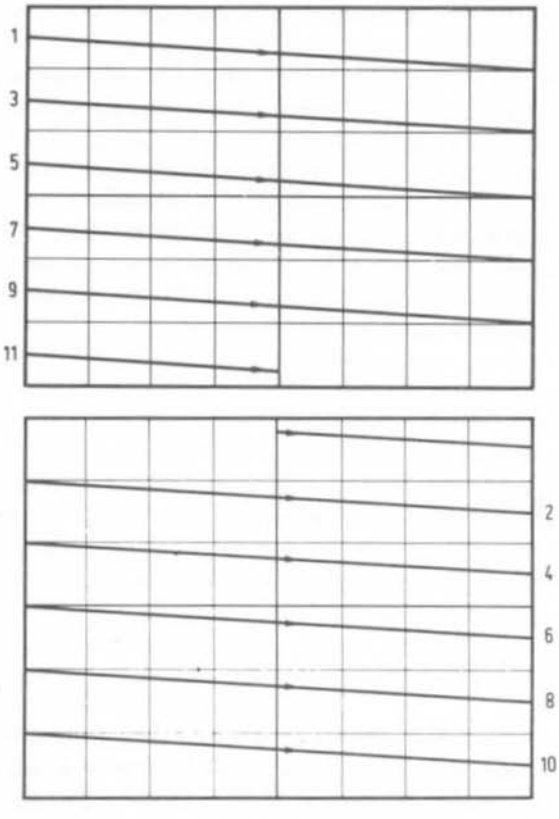


Abb. 7.9.3
Das Zeilensprungverfahren

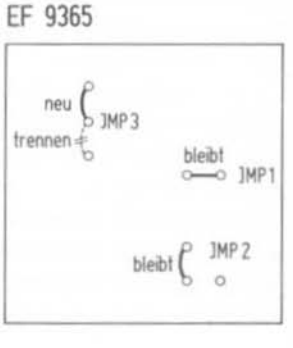


Abb. 7.9.4
Die Brücken
beim Betrieb
mit dem EF9365

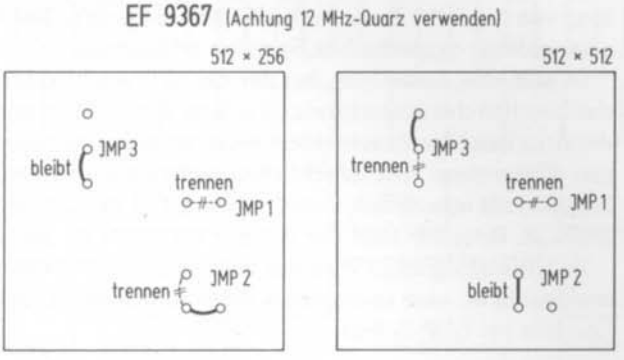


Abb. 7.9.5
Beim EF9367 gibt es zwei verschiedene Kombinationen

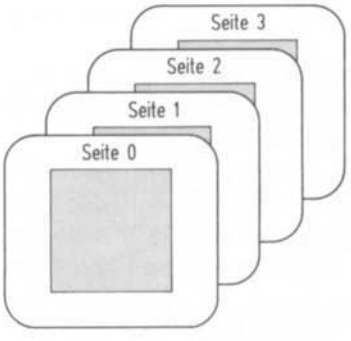
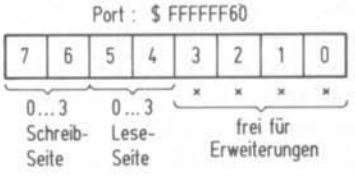


Abb. 7.9.6
Die vier Bildseiten bei der GDP64 mit dem EF9366



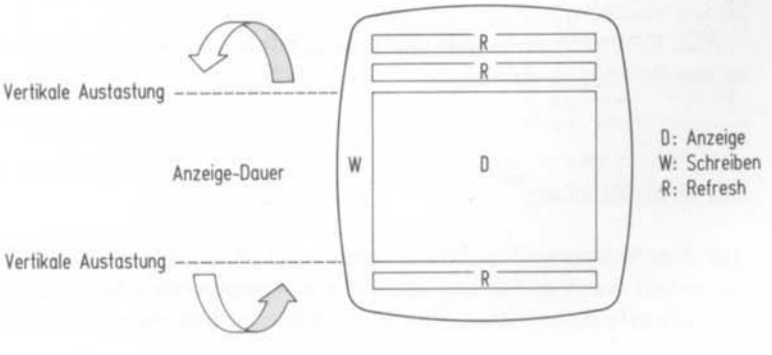


Abb. 7.9.7
Die verschiedenen Zugriffe sind genau aufgeteilt

lung von Schreib-, Lese- und Auffrischvorgängen. Der Refresh ist für die Funktion der verwendeten dynamischen Speicher erforderlich.

Es gibt eine Ausnahme, bei der der GDP doch während der Anzeige-Phase schreibt: das Löschen des Bildschirms. Weichen wir hier von unserem Grundsatz, nur außerhalb der Anzeigephase zu schreiben, ab, so erreichen wir die sehr kurze Löschzeit von nur 20 ms. Wenn man den Löschbefehl verwendet, so muß man die gleiche Lese- und Schreibseite auswählen, sonst wird ein Teil der Leseseite und ein Teil der Schreibseite gelöscht. Betroffen sind die Befehle 4,6,7 und $C der Graphik-Prozessor-Befehle.

Im Grundprogramm wird das automatisch berücksichtigt. Das Unterprogramm CLPG ermöglicht es, eine unsichtbare Seite zu löschen. Dies dauert allerdings länger als das Löschen per GDP-Befehl.

Erzeugung der Steuersignale:

Alle Steuersignale werden von der Quarzfrequenz 14 MHz abgeleitet; sie wird vom Quarzoszillator IC 5 erzeugt. Der Takt wird direkt verwendet, um die einzelnen Bildpunkte aus dem Schieberegister 74LS166 (IC 2) herauszuschieben.

Das IC 3, ein Zähler 74LS163, übernimmt die weitere Signalherstellung. *Abb. 7.9.8* zeigt eine Zusammenstellung der wichtigsten Signale. Der Zähler teilt den Takt zunächst einmal. An QA liegen damit 7 MHz, an QB 3,5 MHz und an QC 1,75 MHz. Der 1,75-MHz-Takt wird auch an den Eingang CK (Pin 1, IC 20) geleitet und bildet den Arbeitstakt des Graphik-Prozessors.

Aus den anderen Signalen werden, durch logische Verknüpfung, die Signale -Ld (Lade-Signal für das Schieberegister), Eingang Pin 15, IC 2, sowie das Signal -RAS und -CAS erzeugt. -RAS gelangt über den Dekoder IC 16 an die RAM-Bausteine; es hat verschiedene Aufgaben. Bei einem Schreibvorgang wählt es nur eine der -RAS-Leitungen des RAMs aus. Bei einem Lese-Vorgang, oder beim Refresh, werden alle -RAS-Leitungen der RAMs gleichzeitig mit dem Signal -QC versorgt. Das Signal -CAS gelangt direkt an die RAMs. Das Signal MUXA liegt am Pin 14, IC 17 an und versorgt die beiden Multiplexer in dem IC . Die Adresse an der Leitung A7 (Pin 9, IC 6 bis IC 12) ist nämlich gemultiplext; sie trägt eigentlich zwei Adressen. Das Signal MUXA gibt an, ob die eine oder die andere der beiden Adressen Gültigkeit erlangen soll und sorgt somit dafür, daß diese beiden Adressen richtig zum RAM übertragen werden. Diese beiden Adressen bilden nämlich die Seitenadresse 0, 1, 2 und 3.

Abb. 7.9.9 zeigt nochmals das Timing mit allen realen Verzögerungszeiten, so wie es an der Baugruppe anliegt.

Der BAS-Mischer:

Aus dem Videosignal und den sogenannten Synchronsignalen muß nun eine Mischung hergestellt werden, die ein Monitor am sogenannten BAS-Eingang versteht. Die Synchronsignale dienen dazu, den Bildrahmen zu bestimmen, also festzulegen, wann ein

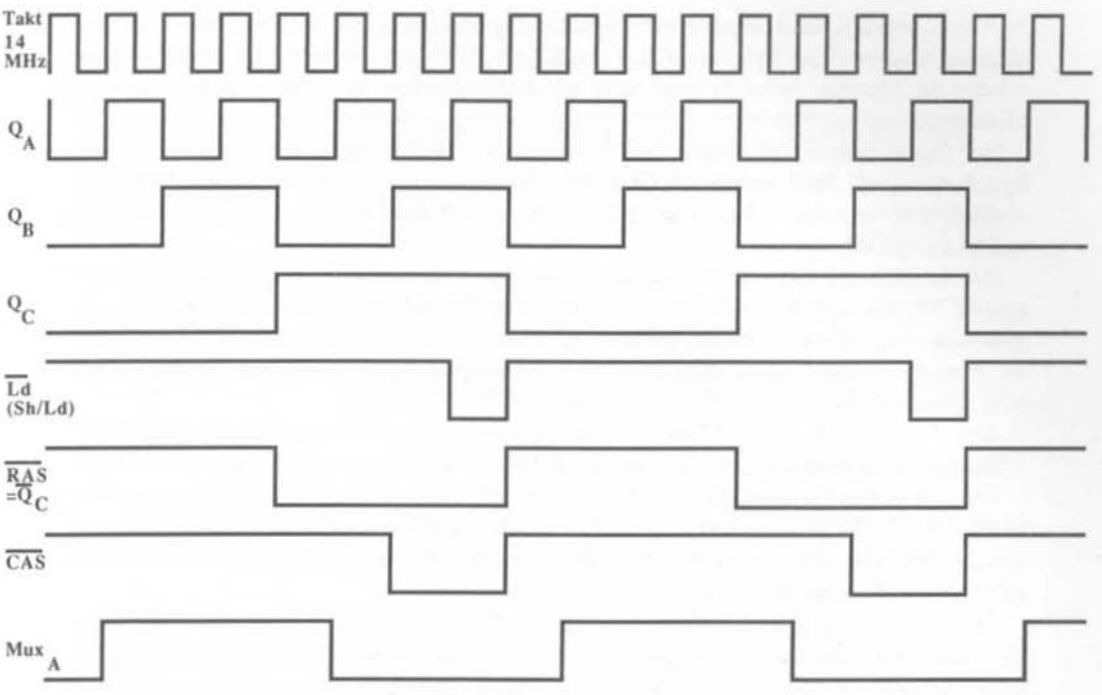


Abb. 7.9.8 Das Timing im Prinzip

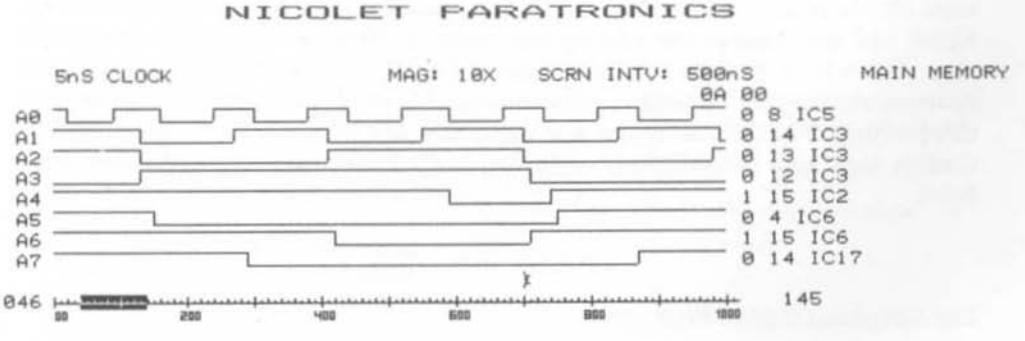


Abb. 7.9.9 Das reale Timing-Diagramm

Bild neu beginnt, und wann eine Bildzeile abgeschlossen ist. Man hat aus Vereinfachungsgründen diese Synchronsignale mit dem Bildsignal vereint und trennt sie erst wieder im Monitor (oder Fernsehgerät mit Video-Buchse) auf. *Abb. 7.9.10* zeigt den Mischvorgang.

Das Signal V-Sync (vertikales Synchronsignal) und das Signal H-Sync (horizontales Synchronsignal) sind bereits im GDP zum Signal SYNC, Pin 34, IC 20 gemischt. Sie werden nun von der Schaltungslogik mit IC 1 und dem Transistor zum BAS-Signal zusammengefaßt.

Die Amplituden der Synchronsignale und des Videosignals sind durch die Widerstände R3, R5 und R7 bestimmt. Die Widerstände wurden so gewählt, daß sich ein genormtes Signal ergibt. Wenn das Bild mit diesem Signal auf manchen Monitoren aber zu dunkel erscheint, kann man den Widerstand R3 einfach weglassen. Es ergibt sich eine höhere Video-Amplitude und ein helleres Bild.

Abb. 7.9.11 zeigt das Schema für die Verschaltung mit einem Monitor. Der BAS-Ausgang (siehe Bestückungsplan) wird direkt mit dem Monitor-Eingang verbunden.

Wenn man ein Fernsehgerät mit AV-Buchse besitzt, kann man auch diese verwenden (*Abb. 7.9.12*). Der Widerstand mit 75 Ω ist nötig, um einen unerwünschten Gleichspannungsanteil, der vom Fernseher kommt, zu unterdrücken. Die Brücke von 1 nach 5 schaltet den Fernseher auf die AV-Buchse um.

Abb. 7.9.13 zeigt einige weitere gebräuchliche Video-Buchsen. Dabei wird bei manchen ein getrenntes Synchronsignal gefordert, das aus der GDP ebenfalls herausgeführt ist. Das Signal der GDP ist aber mit einem Offen-Kollektor-Ausgang versehen, den man zuvor über einen Widerstand nach +5 V verbinden muß. Manchmal ist die Polarität nicht korrekt, und man muß ggf. einen Inverter dazwischenschalten. Das VB-Synchron-Signal ist übrigens nicht mit dem im SYNC-Signal enthaltenen V-SYNC identisch, sondern viel länger. Das führt bei manchen Geräten mit getrennten Synchronsignalen zu Verzerrungen im oberen Bildteil. Sie müssen dann das Signal mit Hilfe eines Monoflops verkürzen. Vermeiden Sie aber möglichst die Verwendung der getrennten Signale, und verwenden Sie einen BAS Eingang, dann gibt es keine Probleme.

Ein anderer Weg, das Signal auf einen Bildschirm zu bringen, ist die Verwendung eines HF-Modulators. *Abb. 7.9.14* zeigt ein Schema. Der HF-Modulator erhält das BAS-Signal und eine zusätzliche Versorgungsspannung. Meist genügen +5 V, manchmal aber sind +12 V notwendig. Der Ausgang des HF-Modulators wird dann an den Antenneneingang des TV-Gerätes angeschlossen. Man muß dann über die Kanalschalter den Sender (HF-Modulator) finden und abgleichen. Die Bildqualität ist generell bei TV-Geräten nicht sehr befriedigend, weshalb sich die Verwendung eines Monitors empfiehlt.

Der Graphik-Display-Prozessor:

Abb. 7.9.15 zeigt den GDP-Baustein. Wir wollen uns jetzt jedoch nicht weiter in die Schaltungtechnik vertiefen, sondern uns den inneren Funktionen des Bausteins widmen.

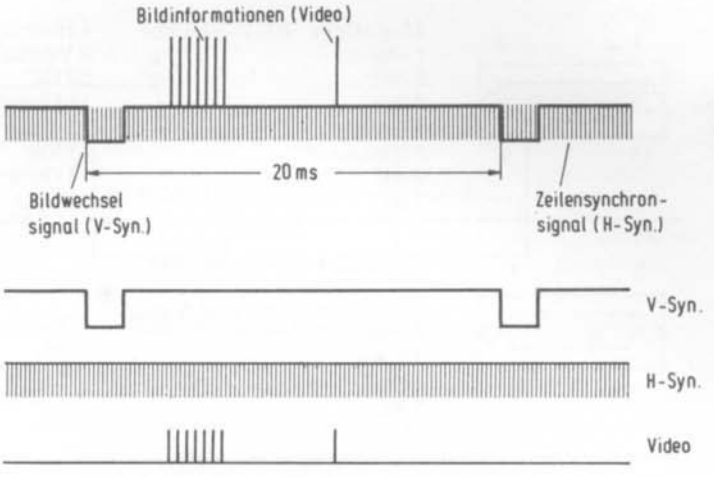


Abb. 7.9.10
Die Mischung des BAS-Signals

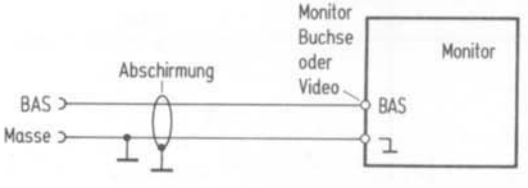


Abb. 7.9.11
Anschluß eines Monitors

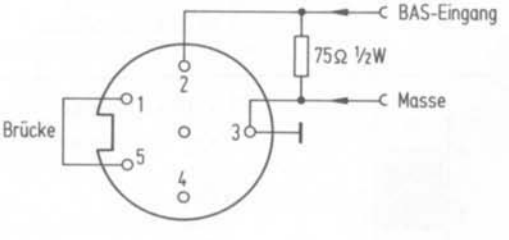
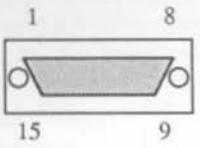
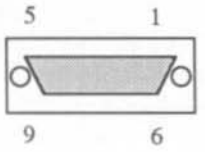


Abb. 7.9.12
Der Anschluß an eine AV-Buchse



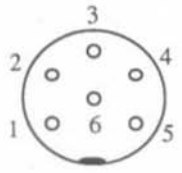
15-poliger RGB-Stecker

1 Masse
2 NC
3 Rot
4 Grün
5 Blau
6 NC
7 NC
8 Horizontal- oder H/V-Signal
9 Vertikalsynchron-Signal
10 NC
11 Masse
12 Ton-Frequenz
13 Rot
14 Grün
15 Blau



9-poliger RGB-Stecker

1 Horizontal- oder H/V-Signal
2 Rot
3 Grün
4 Blau
5 NC
6 Masse
7 NC
8 Vertikalsynchron-Signal
9 Y (NC)

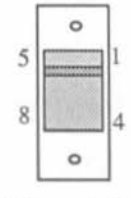


DIN-Stecker (DIN 45482)

1 Schaltspannung
2 Video-Eingang (BAS)
3 Masse
4 Ton-Eingang
5 Versorgungsspannung 12V

Scart-Buchse

1 Audio-Ausgang B, Stereo-Kanal R
2 Audio-Eingang B, Stereo-Kanal R
3 Audio-Ausgang A, Stereo-Kanal L
4 Audio-Masse
5 RGB Blau-Masse
6 Audio-Eingang A, Stereo-Kanal L
7 RGB Blau-Signal
8 Schaltspannung
9 RGB Grün-Masse
10 Datenleitung 2
11 RGB Grün-Signal
12 Datenleitung 1
13 RGB Rot-Masse
14 Reserve
15 RGB Rot-Signal
16 Austastsignal
17 Video-Masse
18 Austastsignal-Masse
19 Video-Ausgang
20 Video-Eingang



Lücke zwischen Pin 1,5 und dem Rest

8-Pin-Buchse

1 NC
2 Rot
3 Grün
4 Blau
5 Masse
6 Masse
7 Horizontalsynchron-Signal
9 Vertikalsynchron-Signal

Abb. 7.9.13
Verschiedene Buchsenformen

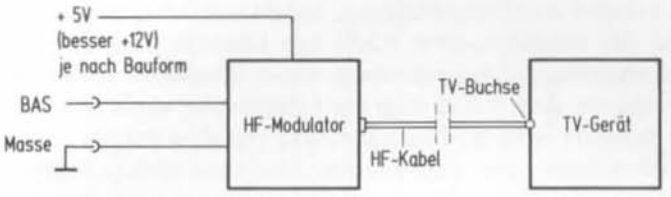


Abb. 7.9.14 Anschluß eines HF-Modulators

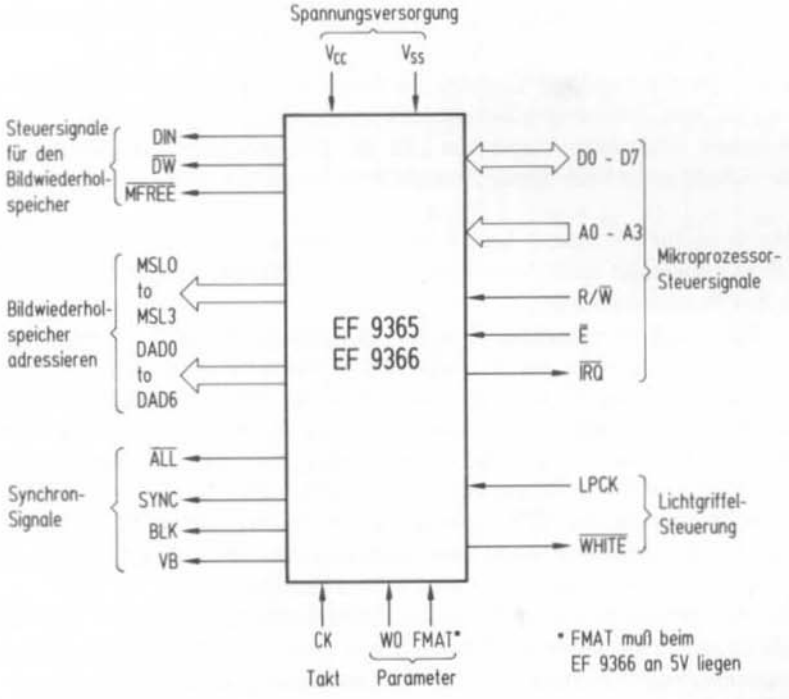


Abb. 7.9.15 Die Signale des EF9365/9366

Der GDP besitzt 16 adressierbare Register; *Abb. 7.9.16* zeigt ein Schema. Die Adressen $FFFFFF70 bis $FFFFFF7F sind den Registern 0 bis $F zugeordnet.

Das Register 0 besitzt eine Doppelfunktion. Beim Lesen ist es das Status-Register und beim Schreiben das Befehls-Register (CMD von Command, der Befehl). Im Status-Register werden die aktuellen Informationen, wie z. B. eine Bereitmeldung zum nächsten Befehl, übergeben. *Abb. 7.9.17* zeigt die Belegung der einzelnen Status-Bits.

Bit 0 dieses Registers wird im Zusammenhang mit dem Betrieb eines Lichtgriffels benötigt. Die GDP64-Baugruppe ist für den Anschluß eines solchen Griffels ausgestattet. Beim Betrieb dieser Einrichtung wird zunächst eine sogenannte Lichtgriffel-Sequenz durch einen speziellen Befehl gestartet; das Ende der Sequenz ist dann an diesem Bit ablesbar. Bit 0 liegt auf 1, wenn die Licht-Griffel-Sequenz beendet wurde, und damit liegt dann in den Registern XLP und YLP eine gültige Information.

Bit 1 ist identisch mit dem Ausgang VB; man kann dort den 20-ms-Takt des Bildwechsels abfragen.

Bit 2 ist auf 1, wenn der GDP bereit ist, einen neuen Befehl zu empfangen. Gibt man einen Befehl an den GDP, so braucht dieser natürlich eine Weile, bis er ihn ausgeführt hat. Das Bildschirmlöschen benötigt z. B. 20 ms. In dieser Zeit liegt dann an Bit 2 eine 0 an. Das Unterprogramm „WAIT" im Grundprogramm wartet so lange, bis dieses Bit wieder auf 1 liegt.

Bit 3 liegt immer dann auf 1, wenn der Schreibstift das Bildschirmfenster verlassen hat. In den X- und Y-Registern kann man Koordinaten im Bereich 0 bis 4095 unterbringen. Bei einem Bildschirmformat von 512 mal 256 ist aber nur ein kleiner Ausschnitt sichtbar. Sobald außerhalb dieses Ausschnittes gearbeitet wird, erscheint an Bit 3 eine 1.

Die Werte an Bit 4 bis Bit 7 dienen der Interruptverarbeitung, die bei uns normalerweise nicht verwendet wird. Dazu müßte man den IRQ-Ausgang des GDP mit der INT-Leitung des Busses verbinden.

Abb. 7.9.18 zeigt die möglichen Befehle des Graphik-Display-Prozessors. Wir besprechen zunächst die Codes im Bereich $20 bis $7F. Sie entsprechen den ASCII-Werten für die Zeichendarstellung. Wenn man einen Wert in diesem Bereich in das Register 0 (Adresse $FFFFFF70) schreibt, so erscheint ein Zeichen auf dem Bildschirm. Die Position des Zeichens ist durch die X- und Y-Register bestimmt, die Größe durch das Register CSIZE und die Richtung durch das Register CTRL2.

Abb. 7.9.19 zeigt den im GDP implementierten Zeichensatz in der Rasterdarstellung. Auffällig ist das Fehlen der deutschen Sonderzeichen; sie werden im Bedarfsfall vom Grundprogramm durch Graphik-Befehle extern gebildet.

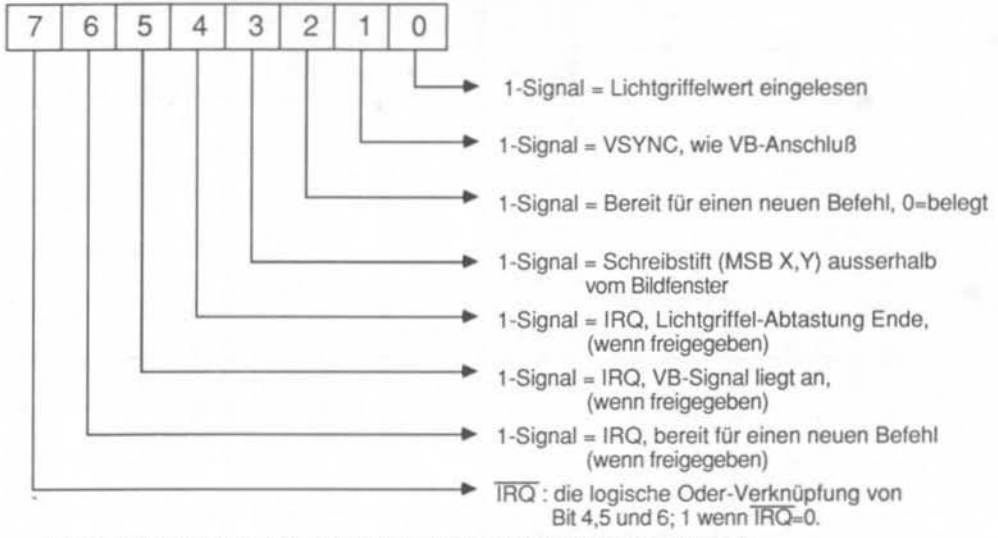
Die Befehle $80 bis $FF sind die sogenannten Kurzvektoren. Dabei wird ein Vektor, ausgehend von der aktuellen X-, Y-Koordinate, gezeichnet. Der Befehl wird aus einzelnen Elementen zusammengesetzt: dem Richtungscode, an den Bitpositionen 0 bis 2, sowie den Distanzen ΔY und ΔX, die an die Positionen Bit 3 und Bit 4, sowie Bit 5 und Bit 6 geschrieben werden. Bit 7 liegt immer auf 1.

Ein Kurzvektor kann maximal 3 Punkte lang sein. Die Ausführung ist aber sehr schnell und im wesentlichen nur von der Geschwindigkeit der Übertragung zum GDP abhängig. Der GDP ist in der Lage, Vektoren mit bis zu 1.5 Millionen Bildpunkten pro Sekunde zu zeichnen.

Adresse beim 68k-System	Adreß-Register					Register-Funktionen		Anzahl der Bits
	Binary				Sed.	Lesen $R/\overline{W} = 1$	Schreiben $R/\overline{W} = 0$	
	A3	A2	A1	A0				
$FFFFFF70	0	0	0	0	0	STATUS	CMD	8
$FFFFFF71	0	0	0	1	1	CTRL 1 (Steuerung von Schreiben und Interrupt)		7
$FFFFFF72	0	0	1	0	2	CTRL 2 (Vektor- u. Zeichengenerator-Steuerung)		4
$FFFFFF73	0	0	1	1	3	CSIZE (Zeichengröße)		8
$FFFFFF74	0	1	0	0	4	Reserviert		–
$FFFFFF75	0	1	0	1	5	DELTAX		8
$FFFFFF76	0	1	1	0	6	Reserviert		–
$FFFFFF77	0	1	1	1	7	DELTAY		8
$FFFFFF78	1	0	0	0	8	X MSBs		4
$FFFFFF79	1	0	0	1	9	X LSBs		8
$FFFFFF7A	1	0	1	0	A	Y MSBs		4
$FFFFFF7B	1	0	1	1	B	Y LSBs		8
$FFFFFF7C	1	1	0	0	C	XLP (Lichtgriffel)	Reserviert	7
$FFFFFF7D	1	1	0	1	D	YLP (Lichtgriffel)	Reserviert	8
$FFFFFF7E	1	1	1	0	E	Reserviert		–
$FFFFFF7F	1	1	1	1	F	Reserviert		–

Abb. 7.9.16 Die Register des GDPs

STATUS (Register $FFFFFF70) Nur Lesen



Bit 0,1 und 2 werden nicht zwischengespeichert und nicht ausgeblendet
Bit 4,5 und 6 werden auf Null gesetzt, nachdem dieses Status-Register gelesen wurde

Abb. 7.9.17 Das Status-Register

b3 b2 b1 b0		b7 b6 b5 b4 = 0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0 0 0 0	0	Setzt Bit 1 von CTRL 1: Schreibstift	Vektorerzeugung b2, b1, b0 wie bei den Kurz-Vektoren	SPACE	0	@	P	`	p								
0 0 0 1	1	Löscht Bit 1 von CTRL 1: Löschstift		!	1	A	Q	a	q								
0 0 1 0	2	Setzt Bit 0 von CTRL 1: Stift runter		"	2	B	R	b	r								
0 0 1 1	3	Löscht Bit 0 von CTRL 1: Stift rauf		#	3	C	S	c	s								
0 1 0 0	4	Bildschirm löschen		$	4	D	T	d	t								
0 1 0 1	5	X,Y-Register auf 0		%	5	E	U	e	u								
0 1 1 0	6	X,Y-Register auf 0 und Bildschirm löschen		&	6	F	V	f	v								
0 1 1 1	7	Bildschirm löschen, CSIZE auf $11, alle Register außer XLP, YLP auf 0		´	7	G	W	g	w								
1 0 0 0	8	Lichtgriffel-Start $\overline{\text{WHITE}}$ auf 0	Spezial-Richtungs-Vektoren b2, b1, b0 wie bei den Kurz-Vektoren	(	8	H	X	h	x								
1 0 0 1	9	Lichtgriffel-Start		)	9	I	Y	i	y								
1 0 1 0	A	5×8 Block zeichnen, je nach CSIZE		*	:	J	Z	j	z								
1 0 1 1	B	4×4 Block zeichnen, je nach CSIZE		+	;	K	[	k	{								
1 1 0 0	C	Bildschirm hell oder dunkel, je nach Stift		,	<	L	\	/	¦								
1 1 0 1	D	X-Register auf 0		–	=	M	]	m	}								
1 1 1 0	E	Y-Register auf 0		.	>	N	↑	n	¬								
1 1 1 1	F	$\overline{\text{MFREE}}$-Signal für Direkt-Zugriff (nicht verwendbar!)		/	?	O	←	o	▒								

Kurz-Vektoren

b7	b6 b5	b4 b3	b2 b1 b0
1	|ΔX|	|ΔY|	Richtung

Dimension

ΔX od. ΔY		Vektor-Länge
0	0	0 Schritte
0	1	1 Schritt
1	0	2 Schritte
1	1	3 Schritte

Richtung

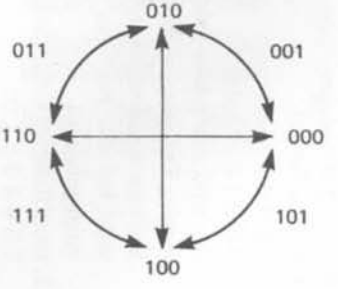


Abb. 7.9.18 Die Befehle des GDPs

Neben den Kurzvektoren gibt es noch die Standard-Vektoren, die den Codebereich $10 bis $1F belegen. Dort gibt es einige Unterschiede.

1. Grund-Befehl:

   7 6 5 4 3 2 1 0
   0 0 0 1 0 dys dxs 1

   dys und dxs geben dabei das Vorzeichen der Register DELTAX und DELTAY an. In diesen Registern steht ein Wert zwischen 0 und 255, der maximalen Länge eines Vektors. Zusammen mit dem Vorzeichen wird die Zeichenrichtung bestimmt. Dabei wird bei der Position x und y begonnen, die durch die entsprechenden Register vorgegeben ist.

2. modifizierter Grundbefehl

   7 6 5 4 3 2 1 0
   0 0 0 1 0 b a 0

   Für b und a gilt folgendes:

   b a
   0 0 DELTAY wird ignoriert, DELTAX>0
   0 1 DELTAX wird ignoriert, DELTAY>0
   1 0 DELTAX wird ignoriert, DELTAY<0
   1 1 DELTAY wird ignoriert, DELTAX<0

   Damit kann man schnell Rechtecke etc. konstruieren, ohne die Register DELTAX oder DELTAY neu belegen zu müssen.

3. Spezial-Grundbefehl

   7 6 5 4 3 2 1 0
   0 0 0 1 1 d2 d1 d0

   d2,d1 und d0 sind so kodiert, wie der Richtungscode bei den Kurzvektor-Befehlen. Das Register DELTAX oder DELTAY mit dem kleineren Inhalt von beiden wird jedoch ignoriert. Als DELTA-Wert wird dann für X und Y der größere von beiden Inhalten eingesetzt. Damit kann man z.B. auch Diagonale etc. leicht konstruieren.

   Letzlich bleiben noch die Codes im Bereich 0 bis ‚F. Sie haben eine sehr unterschiedliche Bedeutung. So gibt es z. B. Befehle zum Löschen des Bildschirms und zum Verändern von Registern.

Der Code 0 setzt den Schreibstift; alle nachfolgenden Zeichenbefehle schreiben dann hell auf dunkel. Durch den Code 1 wird bewirkt, daß alle nachfolgenden Zeichenbefehle dunkel auf hell schreiben. Dabei hängt es vom vorhandenen Bild ab, ob man den einen oder anderen Vektor sieht. Schreibt man dunkel auf hell, so muß natürlich auch der entsprechende Bildteil vorher hell gewesen sein.

Mit dem Code 2 wird der Stift gesenkt, dann erst finden überhaupt Schreiboperationen statt. Mit dem Code 3 kann man den Stift heben, es erfolgt dann keine Schreiboperation mehr.

Neben diesen Befehlen, die an das Register 0 gegeben werden, gibt es weitere Steuerregister.

Abb. 7.9.20 zeigt die Belegung des Registers CTRL 1 (0 oder Adresse $FFFFFF71).

Bit 0 und 1 bestimmen die Schreibart. Diese beiden Bits werden z.B. auch durch die Befehle mit den Codes 0, 1, 2 und 3 beeinflußt. Zum normalen Schreiben mit hell auf dunkel steht dort jeweils der Wert 1.

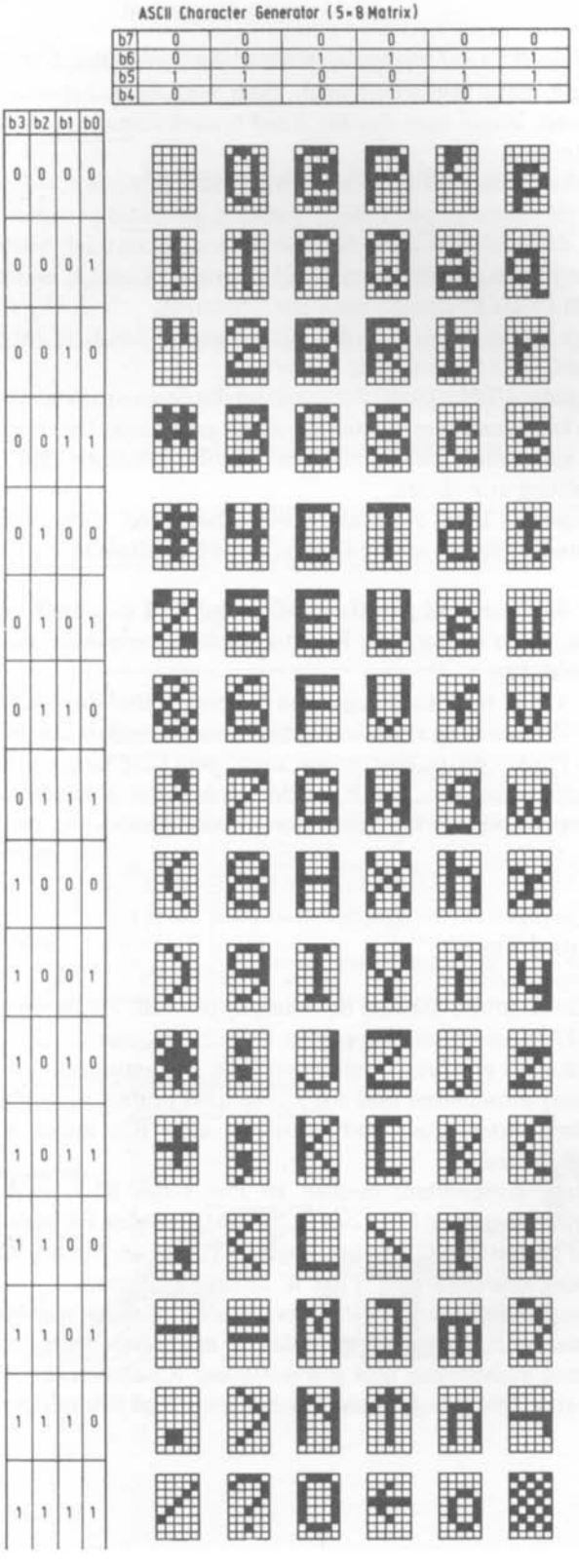


Abb. 7.9.19
Der eingebaute Zeichengenerator

Wenn man Bit 2 auf 1 setzt, wird der Bildschirm dunkel. Dadurch, daß der GDP keine Darstellung mehr bearbeiten muß, kann man dann Schreibvorgänge ohne Wartezeit durchführen. Wenn man das Bit 2 auf 0 zurücksetzt, erscheint das Geschriebene auf dem Bildschirm.

Durch das Setzen des Bits 3 auf 1 wird die Möglichkeit, außerhalb des Bildschirmfensters zu schreiben, gesperrt. Beim Versuch, eine Linie von 0,0 nach 1000,20 zu ziehen, erscheint diese mehrfach auf dem Bildschirm, da sie nach Austritt auf der rechten Seite, wieder von links ins Bild kommt. Die Register X und Y beinhalten aber dennoch den Wert 1000,20 als Endpunkt nach der Operation.

Mit den Bits 4 bis 6 wird der -IRQ-Ausgang des GDP für verschiedene Ereignisse freigegeben.

Das Register CTRL2 (*Abb. 7.9.21*) dient der Steuerung von Vektoren und Zeichen. Für Vektoren kann man hier einstellen, ob sie gepunktet, liniert etc. erscheinen sollen. Bei Zeichen kann man zwischen zwei Schriftrichtungen und zwischen Normal- und Schrägstellung auswählen.

Das Register CSIZE bestimmt die Zeichengröße. Dabei steht der Wert $11 für die kleinste und 0 für die größte Größe (beide Koordinaten mal 16). *Abb. 7.9.22* zeigt die Belegung.

X- und Y-Register sind jeweils in MSB und LSB aufgeteilt, da man den Wertebereich 0...4095 nicht in einem 8-Bit-Register allein unterbringen kann. *Abb. 7.9.23* zeigt die Registerbelegung.

In *Abb. 7.9.24* sind die Belegungen der Lichtgriffel-Register XLP und YLP dargestellt.

Bei der Verwendung eines Lichtgriffels muß man darauf achten, daß man die Position nur auf 8 Punkte genau bestimmen kann; dem GDP liegen ja keine Informationen über die Punktposition vor. Damit ist der Lichtgriffel also weniger für die Eingabe von Zeichnungen, sondern nur zur Auswahl von Menues etc. geeignet.

Aufbau und Test:

Abb. 7.9.25 zeigt die Lötseite der Baugruppe, *Abb. 7.9.26* die Bestückungsseite, und in *Abb. 7.9.27* ist der Bestückungsplan abgedruckt.

1. Einlöten aller passiven Bauteile und der IC-Fassungen.
2. Spannung einschalten und auf Kurzschluß prüfen.
3. Spannung ausschalten und Einsetzen aller ICs, außer den Speichern und dem Graphik-Prozessor.
4. Spannung einschalten; messen an Pin 8 des IC 5, dort muß ein 14-MHz-Takt vorliegen. Messen an Pin 1 des IC 20, dort muß eine Frequenz von 1.75 MHz anliegen. Messen an Pin 20, IC 20, dort liegen 0 V, und an Pin 40, IC 20 liegen +5 V.
5. Spannung ausschalten und das IC EF9366 einsetzen.
6. Spannung einschalten. An Pin 34 muß eine Pulsfolge nachweisbar sein. Den Monitor kann man nun schon mal anschließen. Es muß sich ein dunkles Bildfenster zeigen.
7. Spannung ausschalten und alle restlichen ICs einsetzen.
8. Einschalten. Nun muß immer noch ein dunkler Bildrahmen sichtbar sein.

CTRL1 (Register $FFFFFF71) Lesen und Schreiben erlaubt

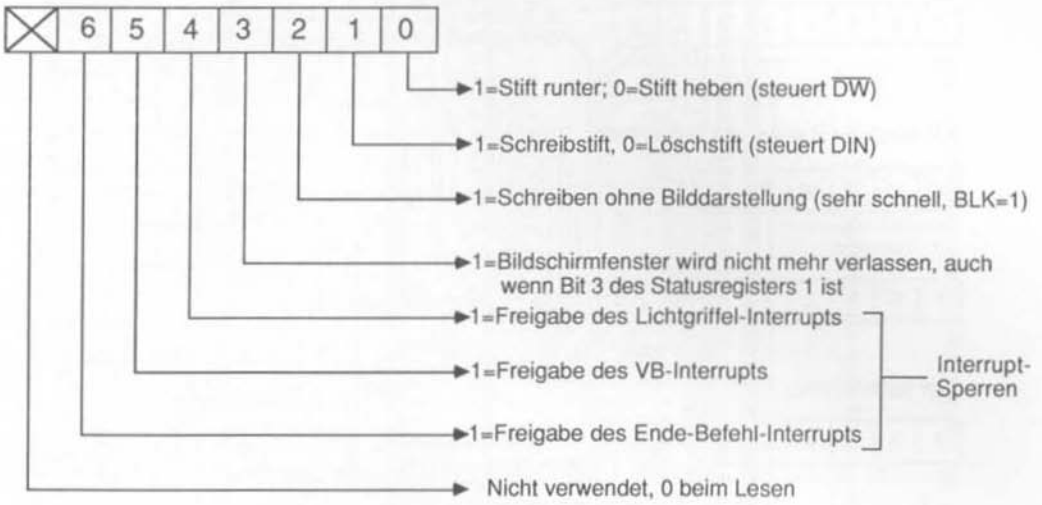


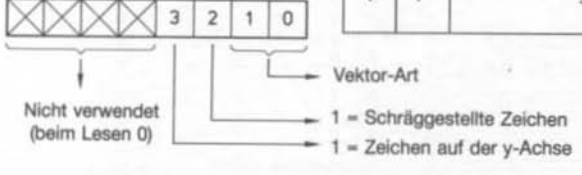
Abb. 7.9.20 Das Register CTRL1

Lesen und Schreiben erlaubt

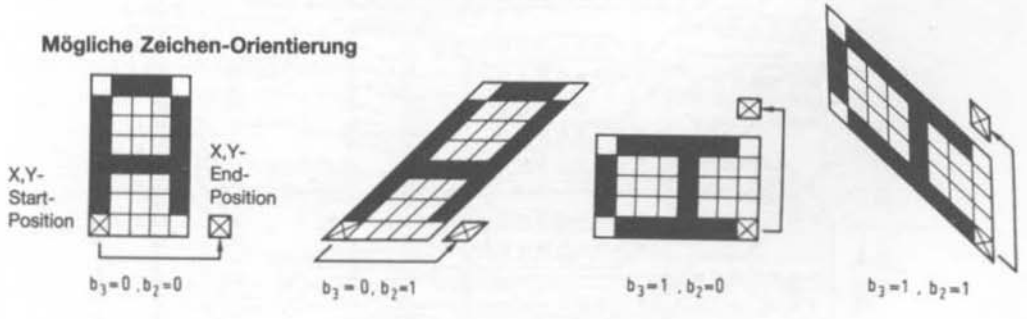
b1	b0	Unterschiedliche Linienarten		
0	0	———	durchgehend	Alle Punkte an
0	1	.....	gepunktet	2 Punkte an, 2 Punkte aus
1	0	- - - - - - -	strichliert	4 Punkte an, 4 Punkte aus
1	1	—·—·—·—	gestrichpunktet	10 Punkte an, 2 Punkte aus 2 Punkte an, 2 Punkte aus

Abb. 7.9.21
Das Register CTRL2

CTRL2 (Register $FFFFFF72)



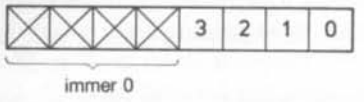
Mögliche Zeichen-Orientierung



CSIZE (Register $FFFFFF73) Lesen und Schreiben erlaubt

P: Skalierung auf der x-Achse (0 = 16, 1 = 1...)
Q: Skalierung auf der y-Achse (0 = 16, 1 = 1...)

X,Y-Register (Lesen und Schreiben)

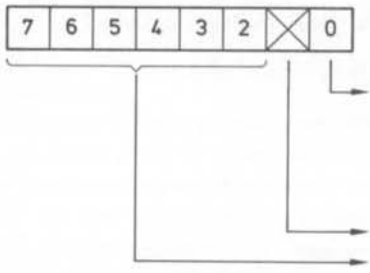


	X	Y
MSB	$FFFFFF78	$FFFFFF7A

LSB $FFFFFF79 $FFFFFF7B

XLP, YLP (nur Lesen)

XLP ($FFFFFF7C)



YLP ($FFFFFF7D)

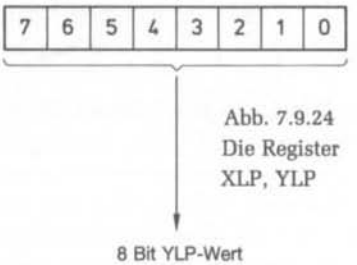


Abb. 7.9.24
Die Register
XLP, YLP

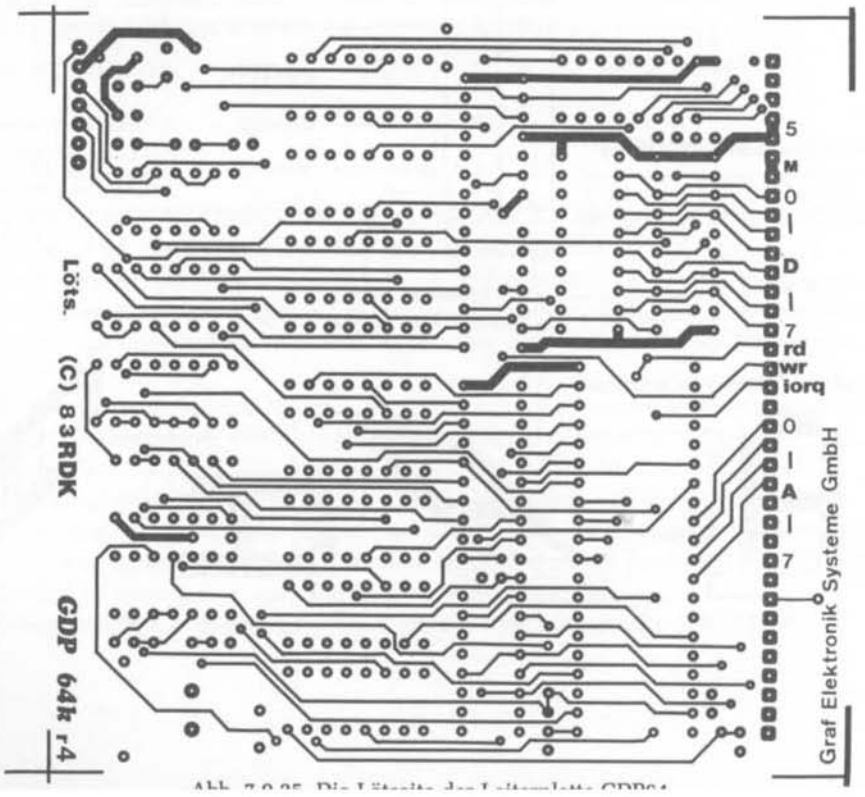
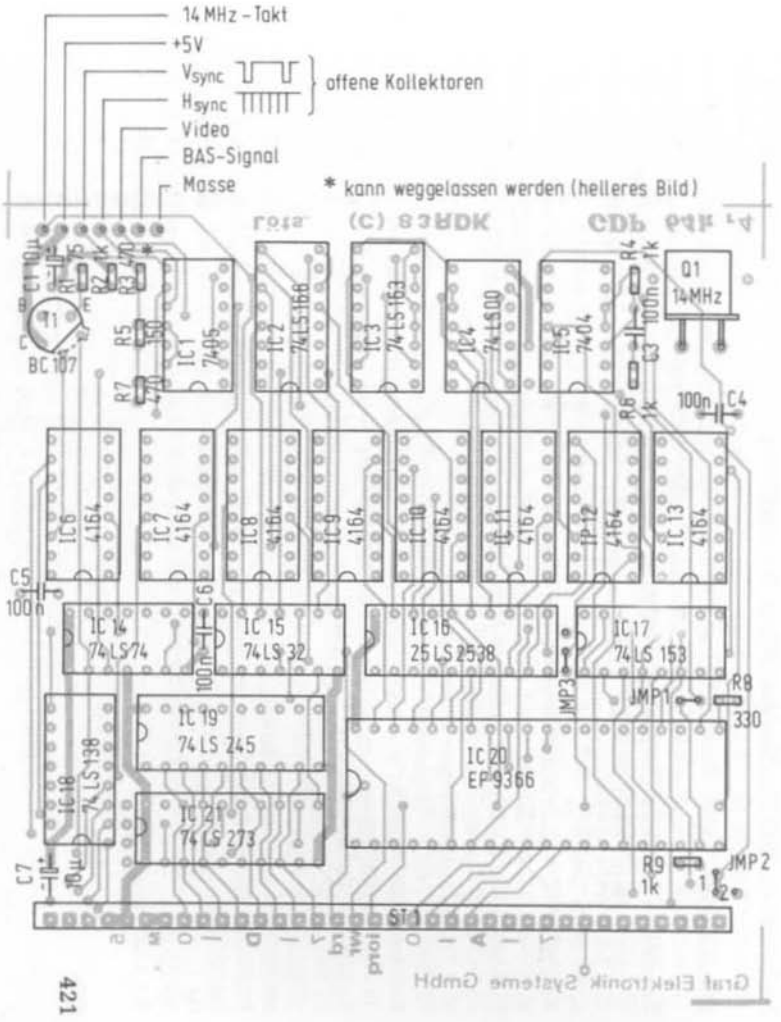


Abb. 7.9.25 Die Lötseite der Leiterplatte GDP64



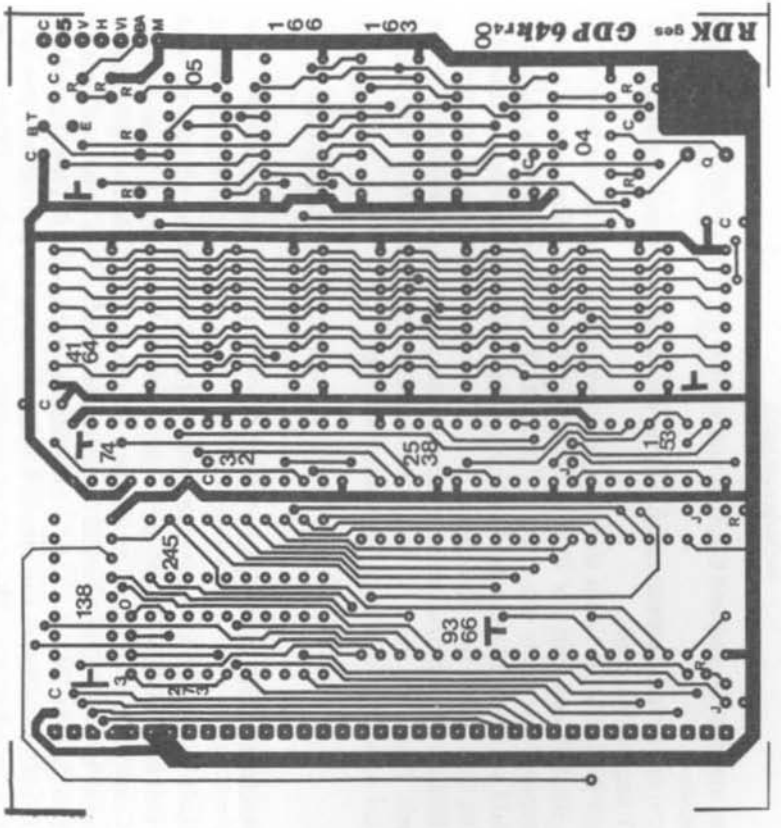


Abb. 7.9.26 Die Bestückungsseite der Leiterplatte GDP64

Links: Abb. 7.9.27 Der Bestückungsplan der Baugruppe GDP64

9. Ausschalten und die Baugruppen CPU68K und ROA64 mit dem Grundprogramm auf den Bus setzen.
10. Einschalten; das Grundprogramm muß sich auf dem Bildschirm melden (auch wenn keine KEY angechlossen ist).

Hinweise zur Fehlersuche: Wenn sich das Grundprogramm nicht meldet, so kann es entweder an der CPU68K mit der ROA64 liegen, oder an der Dekodierung auf der GDP64-Baugruppe. Messen Sie den Eingang Pin 17, IC 20, dort müssen Pulse anliegen.

Wenn im Bild senkrechte Linien fehlen, so liegt dies an einem Speicher. Prüfen Sie den Q-Ausgang (Pin 14) an den Speichern.

Wenn das Bild zerstört ist bzw. nicht erkennbar, messen Sie die Adreßleitungen der Speicher durch. Pegel, die um 1 V liegen, deuten einen Kurzschluß an. Also immer dann, wenn auf dem Skop drei Pegel erkennbar werden.

7.10 Die Cassettenschnittstelle CAS

Eine Möglichkeit unter vielen, Programme und Daten dauerhaft zu speichern, wird durch einen handelsüblichen Cassettenrekorder geboten. Um einen solchen an den NDR-KLEIN-Computer anschließen zu können, ist die hier beschriebene Baugruppe notwendig. Dabei hat die Baugruppe die Aufgabe, die Daten in ein serielles Signal zu verwandeln. Das Signal wird dann für die Aufzeichung aufbereitet. Bei der Wiedergabe muß das vom Cassettenrekorder kommende Signal wieder in ein brauchbares Datensignal zurückverwandelt werden. Die CAS arbeitet mit einer Baudrate von 1200 Baud, kann also ca. 100 Zeichen pro Sekunde verarbeiten.

Materialliste:

IC 1 CD 4070 CMOS-Exklusiv-Oder-Glieder
IC 2 CD 4047 CMOS-Einfach-Monoflop
IC 3 74LS74 D-Flip-Flop
IC 4 NE555 Timer
IC 5 MC 6850 ACIA, Parallel/Seriell/Parallel Umsetzer
IC 6 74LS00 Nand-Glieder
IC 7 MC14528/CD4528 CMOS-Doppel-Monoflop
IC 8 CA3130 Operationsverstärker
IC 9 74LS245 bidirektionaler Bustreiber
IC 10,IC 11 74LS85 Vergleicher
1x 24polige IC-Fassung
1x 20polige IC-Fassung
3x 16polige IC-Fassung
4x 14polige IC-Fassung
2x 8polige IC-Fassung
R1,R5,R6 10 kΩ 1/8 Watt (unkrit.)
R2,R3,R7,R10,R11 1 kΩ
R8 100 kΩ
R4 1.2 kΩ
R9 2.2 kΩ
Tr1 1 kΩ Helitrimmer

Tr2 5 kΩ Helitrimmer
C1 100 nF, keine keramische Scheibe
C2 10 nF
C3 100 nF, keine keramische Scheibe
C4,C5 2.2 nF
C6,C9 100 nF
C7 47 pF
C8 10 µF
D1,D2 Si-Dioden (1N4148)
Stecker1 36polige, gewinkelte Stiftleiste
1x Leiterplatte CAS
1x Dioden-Buchse
2x Einpolige Ein-Schalter

Kenndaten:

Spannungsversorgung: +5 V, Stromaufnahme: 200 mA

Abb. 7.10.1 zeigt den Schaltplan der CAS-Baugruppe. Die Dekodierung übernehmen die beiden Vergleicher IC 10 und IC 11. Die Adresse kann man zwar prinzipiell auf der Lötseite umstellen, jedoch ist sie normalerweise auf $FFFFFFCA und $FFFFFFCB fest eingestellt. Der Baustein MC6850 besitzt zwei IO-Port-Adressen, die über den Eingang RS eingestellt werden.

Die Adresse $FFFFFFCA spricht beim Schreiben das sogenannte Control-Register an, die Belegung ist in *Abb. 7.10.2* dargestellt. *Abb 7.10.3* zeigt die Belegung des Status-Registers, dessen Inhalt von der Adresse $FFFFFFCA gelesen werden kann. Unter der Adresse $FFFFFFCB wird der Datenport (Lesen und Schreiben) angesprochen.

Der Baustein MC6850 ist ein sogenannter UART (oder ACIA), der die Aufgabe hat, Daten in ein serielles Format umzuwandeln. *Abb. 7.10.4* zeigt ein Schema. Es handelt sich dabei um ein sogenanntes asynchrones Format. Dabei wird immer zunächst ein Start-Bit ausgesendet, das den Wert 0 hat. Daran kann der Empfänger den Beginn der Übertragung eines Bytes erkennen. Dann folgen nacheinander die Datenbits, beginnend mit Bit 0, dann Bit 1 usw., hier bis Bit 7. Es werden also 8 Datenbits übertragen. Danach folgt, vor der Übertragung des nächsten Bytes, das immer auf eins liegende Stop-Bit.

Diese seriellen Daten kann man nicht direkt zu einem Cassettenrekorder übertragen, man muß sie modulieren. Das geschieht hier durch das IC 1 im Bereich der Pins 8, 9, 10. Durch dieses Exklusiv-Oder-Glied werden die Daten mit dem Sendetakt moduliert; er beträgt 1200 Hz.

Dieser Takt bestimmt auch die Übertragungsrate, die man in Baud mißt. Sie beträgt bei der durch das Grundprogramm eingestellten, Betriebsart 1200 Baud. Damit werden 1200 Bits pro Sekunde übertragen, das entspricht etwa 100 Zeichen pro Sekunde.

Abb. 7.10.5 zeigt das Mischergebnis hinter dem Exklusiv-Oder-Glied. Dieses Signal wird, immer wenn der Schalter S1 geschlossen ist, zum Cassettenrekorder übertragen.

Der Takt wird durch den Baustein NE555 (IC 4) erzeugt, der einen besonders stabilen Takt liefert. Der Baustein erzeugt einen Takt von 2400 Hz, der durch den Trimmer Tr1 eingestellt werden muß. Der Kondensator C3 ist kritisch, er muß eine gute Qualität besitzen und darf keine keramische Scheibe sein. Am besten eignet sich ein Wickelkondensator mit hoher Stabilität.

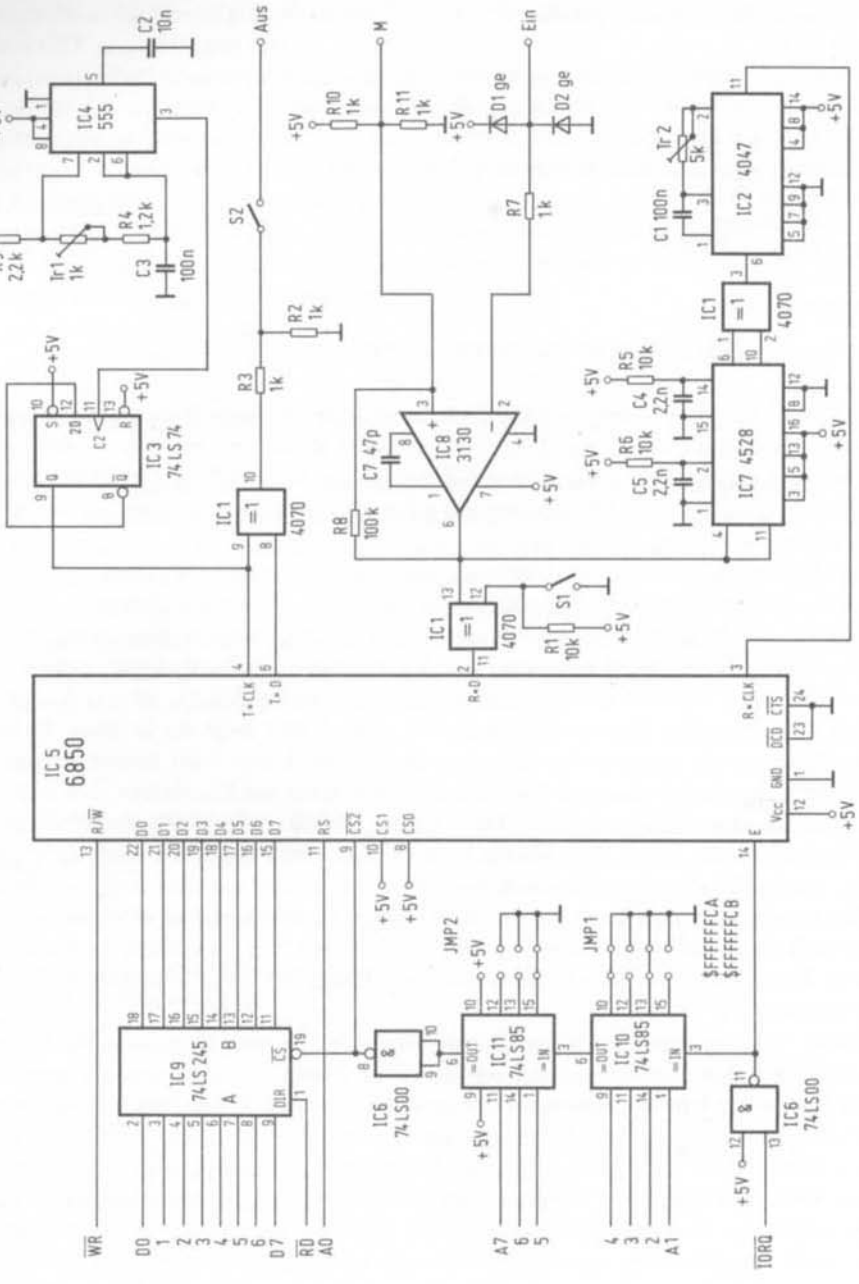


Abb. 7.10.1 Das Schaltbild der Baugruppe CAS

Control Register Adresse $FFFFFFCA (schreiben)

```
7   6   5   4   3   2   1   0

|   |   |   |   |   |   0   0     /1
|   |   |   |   |   |   0   1     /16
|   |   |   |   |   |   1   0     /64
|   |   |   |   |   |   1   1     Master Reset
|   |   |   0   0   0   7 Bits + Even Parität + 2 Stop Bits
|   |   |   0   0   1   7 Bits + Odd Parität + 2 Stop Bits
|   |   |   0   1   0   7 Bits + Even Parität + 1 Stop Bit
|   |   |   0   1   1   7 Bits + Odd Parität + 1 Stop Bit
|   |   |   1   0   0   8 Bits + 2 Stop Bits
|   |   |   1   0   1   8 Bits + 1 Stop Bit
|   |   |   1   1   0   8 Bits + Even Parität + 1 Stop Bit
|   |   |   1   1   1   8 Bits + Odd Parität + 1 Stop Bit
|   |   |
|   0   0     -RTS auf Low, Transmitter Interrupt disabled
|   0   1     -RTS auf Low, Transmitter Interrupt enabled
|   1   0     -RTS auf High, Transmitter Interrupt disabled
|   1   1     -RTS auf Low, Ausgabe eines Break-Zeichens
|             (dauerhaft 0 am seriellen Datenausgang) und
|             Transmitter Interrupt disabled
|
0   Receiver Interrupt disabled
1   Receiver Interrupt enabled
```

Abb. 7.10.2 Das Steuer-Register des 6850

Status Register hier $FFFFFFCA (lesen)

```
7   6   5   4   3   2   1   0

|   |   |   |   |   |   |   1=Receive Daten sind da
|   |   |   |   |   |   1=Transmitter ist leer
|   |   |   |   |   -DCD ( Data carrier detect, Träger vorhanden)
|   |   |   |   -CTS (Clear to send, bereit zum Senden)
|   |   |   FE (Framing error, Format Fehler)
|   |   OVRN (Receiver overrun, Überlauf)
|   PE (Parity error, Paritäts Fehler)
IRQ (Interrupt request, Interrupt Anforderung)
```

Abb. 7.10.3 Das Status-Register des 6850

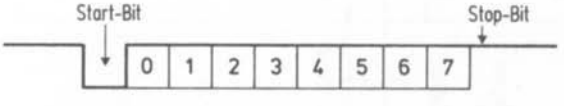


Abb. 7.10.4
Das asynchrone Datenformat

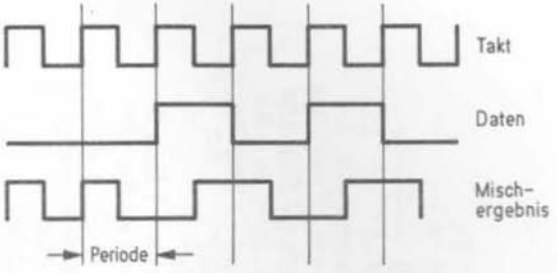
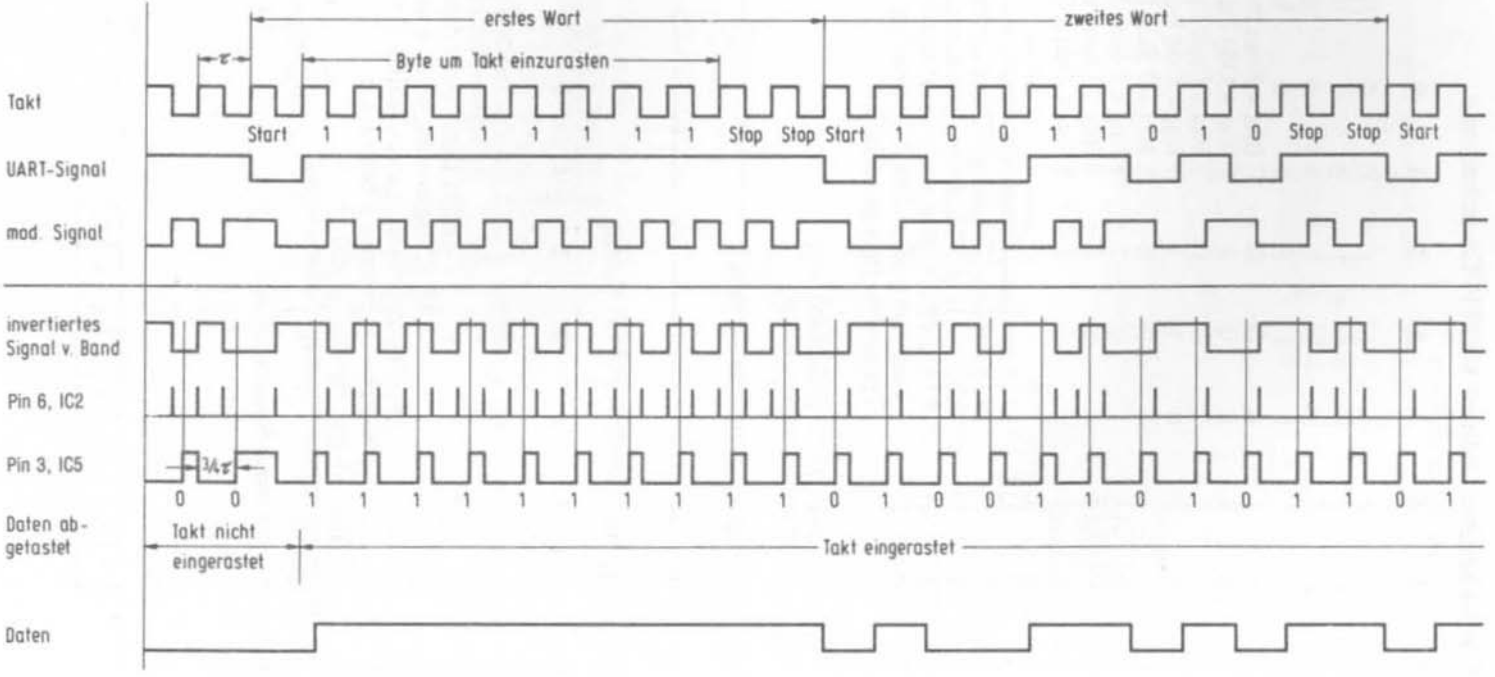


Abb. 7.10.5
So wird das Datensignal moduliert

Abb. 7.10.6
Die Rückgewinnung des Datensignals



Der Takt wird dann durch das Flip-Flip IC 3 durch 2 geteilt, um einen symmetrischen Takt von 1200 Hz zu erhalten.

Das Signal gelangt nun auf den Cassettenrekorder. Das bei der Wiedergabe zurückgelieferte Signal hat keine so steilen Flanken mehr wie bei der Aufzeichnung. Außerdem muß es wieder in das ursprüngliche, serielle Datensignal verwandelt werden, das vom 6850 verarbeitet werden kann. Dazu gelangt es zunächst an den Operationsverstärker IC 8, der es wieder in ein Rechtecksignal verwandelt. Eine Besonderheit der Schaltung ist, daß die Masse des Cassettenrekorders nicht 0 V, sondern ca. 2,5 V hat. Diese Einstellung hat sich als günstig erwiesen. Man muß die Masse des Cassettenrekorders aber auch wirklich mit dem Eingang M verbinden und nicht mit der Masse des Computers, die auf 0 V liegt. Die Masse liegt ferner an Pin 3 des Operationsverstärkers und über R8 am Ausgang des Operationsverstärkers. Damit wird ein Schmitt-Trigger-Effekt erreicht, der für steile Flanken sorgt.

Das Signal gelangt nun zum einen über ein Exklusiv-Oder-Glied (IC 1, Pin 11, 12, 13) an den Eingang RxD des 6850. Das Exklusiv-Oder-Glied ist ferner mit dem Schalter S1 verbunden, mit dem man die Polarität des ankommenden Signals wahlweise invertieren kann. Dies ist nötig, da manche Cassettenrekorder das Signal ebenfalls invertieren. Das Ausgangssignal des Operationsverstärkers gelangt an die Eingänge zweier Monoflops. Dabei reagiert ein Monoflop auf die steigende, das andere auf die fallende Flanke. *Abb. 7.10.6* zeigt ein Schema. An Pin 6, IC 2 kommt dann das Signal beider Monoflops an, das zuvor über ein Exklusiv-Oder-Glied gemischt wurde. Dieses Monoflop muß mit Hilfe des Trimmers TR2 auf eine Zeit, die 3/4 der Periodendauer 833 µs beträgt, eingestellt werden.

Am Ausgang des Monoflops, Pin 3, IC 5, erscheint dann der zurückgewonnene Takt. Dieser Takt ist übrigens erst nach dem ersten übertragenen Zeichen gültig, weshalb man bei Beginn einer Aufzeichnung immer ein zusätzliches Zeichen an den Anfang der Datenfolge stellen muß.

Der 6850 erhält nun einen Takt an RxC und die Daten an RxD. Dabei schadet es hier nicht, daß die Daten noch im modulierten Zustand sind, denn der 6850 fragt den Eingang immer nur bei der steigenden Taktflanke des Signals RxC ab. Wer einmal andere UARTs einsetzen will, muß nachsehen, ob diese auch einen solchen Abtast-Eingang besitzen (z. B. beim 8251 nicht der Fall). GGf. muß man bei solchen UARTs ein D-Flipflop davorsetzen.

Abb. 7.10.7 zeigt den Ausschnitt mit Pulsdiagrammen und *Abb. 7.10.8* das Timing-Diagramm bei einer Datenübertragung; dabei wurde das Datensignal $50 übertragen. Versuchen Sie einmal, diesen Datenwert zu erkennen.

Hinweis: Bei guten Rekordern und Tonbandgeräten kann man die Baudrate wesentlich erhöhen. So sind z.B. 9600 und 7200 Baud möglich. Man muß dazu die Kondensatoren C1 und C3 neu auswählen. Ferner muß man ggf. Die Kondensatoren C4 und C5 mit kleineren Werten, z. B. 1 nF, belegen.

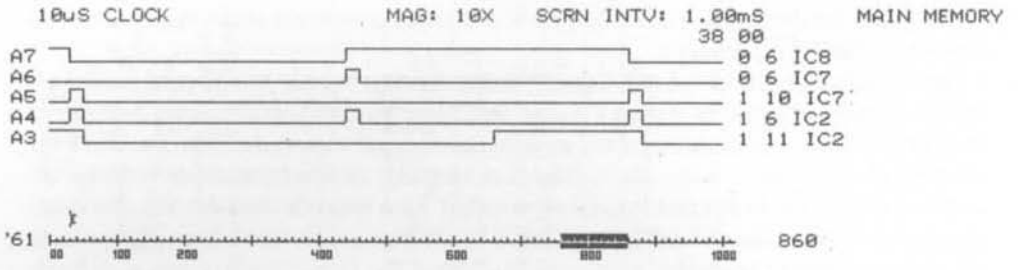


Abb. 7.10.7 Das Timing bei der Rückgewinnung

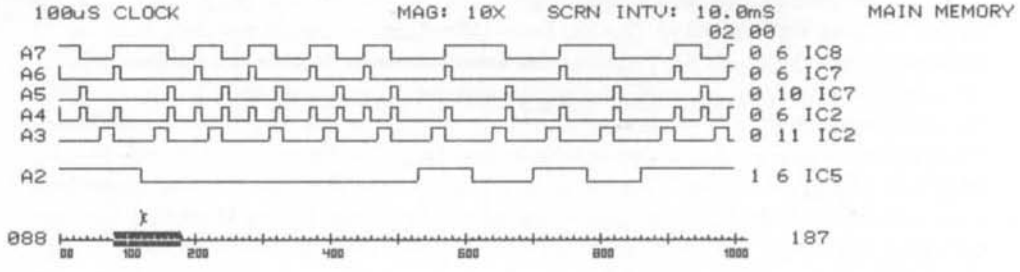


Abb. 7.10.8 Die Übertragung eines Bytes

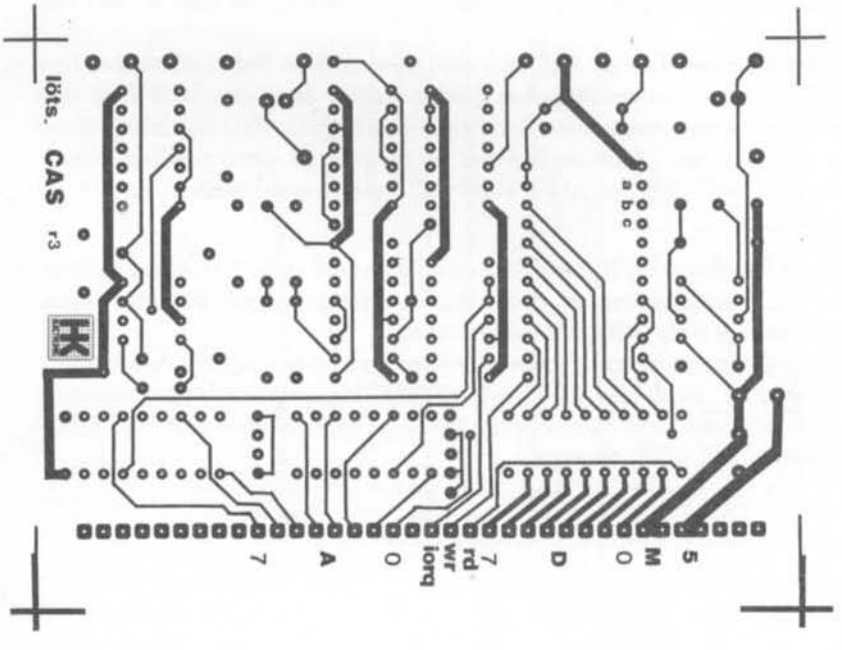


Abb. 7.10.9
Die Lötseite
der Leiterplatte
CAS

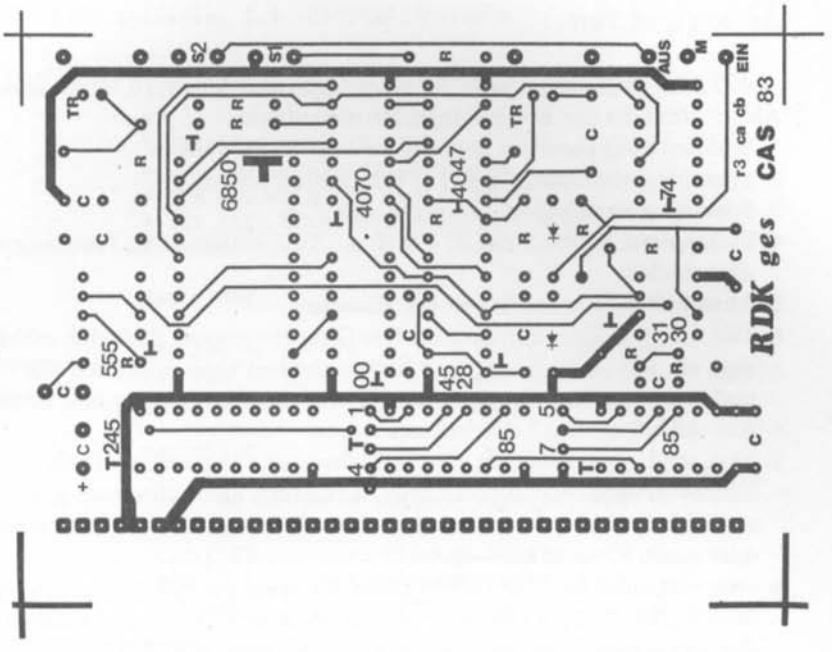


Abb. 7.10.10
Die Bestückungsseite der Leiterplatte CAS

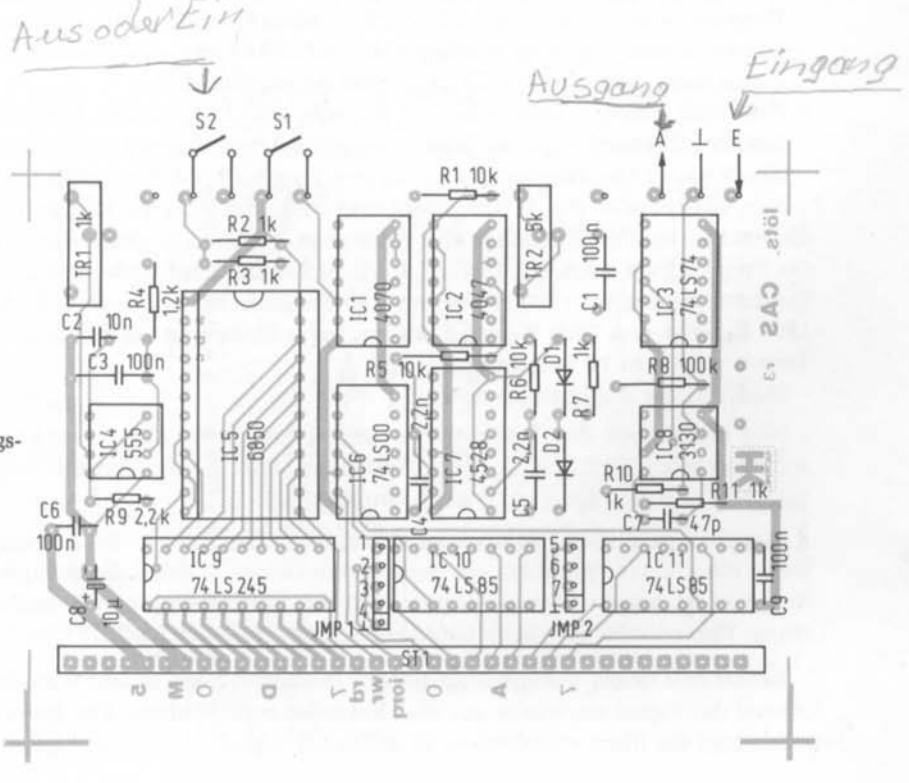


Abb. 7.10.11
Der Bestückungsplan der Baugruppe CAS

Aufbau und Test:

Abb. 7.10.9 zeigt die Lötseite der Baugruppe und *Abb. 7.10.10* die Bestückungsseite. In *Abb. 7.10.11* ist der Bestückungsplan abgedruckt.

1. Einlöten aller passiven Bauteile und der IC-Fassungen.
2. Spannung einschalten und auf Kurzschluß prüfen.
3. Spannung ausschalten, einsetzen von IC 4.
4. Einschalten; an Pin 3 des IC 4 muß ein Takt anliegen (die Frequenz stimmt natürlich noch nicht).
5. Ausschalten, alle restlichen ICs einsetzen.
6. Test mit dem Grundprogramm. Das Grundprogramm muß sich melden; wenn nicht, liegt ein Kurzschluß vor. Zum Kurzschlußtest kann man auch alle ICs herausziehen und dann das Grundprogramm starten. Meldet es sich immer noch nicht, liegt ein Kurzschluß an den Leiterbahnen vor.
7. Nun muß man zunächst die Sendefrequenz abgleichen. Dazu gibt es ein kleines Hilfsprogramm. *Abb. 7.10.12* zeigt das Listing. Man sollte auch gleich den Prüfstecker aus *Abb. 7.10.13* zusammenlöten. Zum Test benötigt man auch eine IOE-Baugruppe oder einen Frequenzmesser.
8. Man verbindet den Port ($FFFFFF30, Bit 0) auf der IOE-Baugruppe mit Pin 4 von IC 5 (6850). *Abb. 7.10.14* zeigt die Position des Anschlusses auf der IOE-Baugruppe, sowie die Einstellung der Brücken für die Adresse $FFFFFF30. Nach dem Start des Programms muß in der oberen Bildschirmhälfte eine Frequenz und in der unteren ein Tastverhältnis angezeigt werden. Das Tastverhältnis beträgt immer 50 %. Die Frequenz kann man nun mit TR1 auf 1200 Hz abgleichen.
9. Nun muß man den Teststecker aus der Abb. 7.10.13 einsetzen. Ferner muß man den Schalter S2 einschalten, so daß das Signal über den Ausgang (CAS Aus) und über den Kondensator des Teststeckers (C) an den Eingang (CAS Ein) geführt wird.

Nun mißt man an Pin 3, IC 5 (6850) und verbindet dazu die Meßleitung der IOE mit diesem Pin. Jetzt muß man mit TR2 abgleichen. Vor dem Abgleich stimmen ggf. sowohl die Frequenz als auch das Tastverhältnis nicht. Zunächst dreht man so lange, bis die Frequenz wieder ca. 1200 Hz beträgt. Die Frequenz springt dabei z.B. von 600Hz auf 1200 Hz oder von 2400 Hz auf 1200 Hz. Dann dreht man vorsichtig so lange, bis das Tastverhältnis zu 75 % angezeigt wird.

Damit ist der Abgleich beendet.

Nun bleibt noch der Anschluß des Cassettenrekoders. *Abb. 7.10.15* zeigt eine Auswahl von gebräuchlichen Steckern. Wenn man keine DIN-Buchse am Cassettenrekorder hat, muß man ggf. die Signale noch anpassen.

Zum Test geht man wie im Kapitel 2.1, nach Abschnitt 4 vor und speichert probeweise ein kleines Programm ab. Beim ersten Laden ist dabei die richtige Schalterstellung von S1 empirisch zu ermitteln. Bei der Verwendung von Fremdkassetten oder einem Rekorderwechsel kann eine Umstellung erforderlich sein.

Leider gibt es bei manchen Rekordern Probleme. Dazu schaut man sich am besten einmal das Signal an, wie es aus dem Rekorder zurückkommt. Für dieses Vorhaben ist unbedingt ein Skop erforderlich. Es sollte das Signal, so wie es aufgezeichnet wurde,

```
Rolf-D.Klein 68000/08 Assembler 4.3 (C) 1984, Seite 1


0E9C00
0E9C00                  *
0E9C00                  * FREQUENZ-MESSGERAET
0E9C00                  * UND TASTVERHAELTNISS-MESSER
0E9C00                  * ZUM ABGLEICH DER CAS-BAUGRUPPE
0E9C00                  * (C) 1985 ROLF-DIETER KLEIN V 1.0
0E9C00                  *
0E9C00
009C00                  ORG $9C00
009C00
= FFFFFF70              GDP EQU $FFFFFF70       * DIREKT ABFRAGEN
= FFFFFF30              IOE EQU $FFFFFF30       * IOE ZUM MESSEN NOTWENDIG
= 00000000              BITNR EQU 0             * MESSEINGANG
009C00
009C00                  FREQMESSE:      * FLANKEN ZAEHLEN
009C00                                  * MESSDAUER 500 MILLISEKUNDEN
009C00  41F9 FFFFFF30    LEA IOE,A0     * INDIREKT ADRESSIEREN
009C06  43F9 FFFFFF70    LEA GDP,A1     * AUCH
009C0C  4281             CLR.L D1       * FREQUENZ MESSEN
009C0E  4242             CLR D2         * VERGLEICHSWERT
009C10  4243             CLR D3         * AUCH BEI GDP
009C12                  *
009C12                  WARTE:          * WARTEN BIS GDP BEREIT
009C12  0811 0001        BTST #1,(A1)
009C16  6700 FFFA        BEQ WARTE      * BIS 1
009C1A                  WARTE0:
009C1A  0811 0001        BTST #1,(A1)   * WARTEN BIS GDP BEREIT
009C1E  6600 FFFA        BNE WARTE0     * BIS 0
009C22  383C 0018        MOVE #25-1,D4  * ANZAHL DER 20 MS MESSVORGAENGE
009C26                  SCHLEIFE:
009C26  1010             MOVE.B (A0),D0
009C28  0240 0001        AND #1,D0
009C2C  B440             CMP D0,D2
009C2E  6700 000A        BEQ SKIP
009C32  3400             MOVE D0,D2
009C34  0681 00000001    ADD.L #1,D1
009C3A                  SKIP:
009C3A  1011             MOVE.B (A1),D0  * GDP BEREIT
009C3C  0240 0002        AND #2,D0       * BIT 1 PRUEFEN
009C40  B043             CMP D3,D0
009C42  6700 FFE2        BEQ SCHLEIFE    * KEINE FLANKE
009C46  3600             MOVE D0,D3      * NEUEN WERT UEBERNEHMEN
009C48  6600 FFDC        BNE SCHLEIFE    * NUR FALLENDE FLANKE INTERESSANT
009C4C  51CC FFD8        DBRA D4,SCHLEIFE  * UND ABZAEHLEN
009C50  4E75             RTS
009C52
009C52                  TIMMESSE:         *
009C52  41F9 FFFFFF30    LEA IOE,A0     * INDIREKT ADRESSIEREN
009C58  4281             CLR.L D1       * ZAEHLER FUER 0-DAUER
009C5A  4282             CLR.L D2       * ZAEHLER FUER 1-DAUER
009C5C                  TWARTE:
009C5C  0810 0000        BTST #BITNR,(A0)
```

zu Abb. 7.10.12

```
Rolf-D.Klein 68000/08 Assembler 4.3 (C) 1984, Seite 2

009C60  6700 FFFA        BEQ TWARTE     * BIS 1
009C64                   *
009C64                   TWARTE0:
009C64  0810 0000        BTST #BITNR,(A0)
009C68  6600 FFFA        BNE TWARTE0    * BIS 0
009C6C                   *
009C6C                   TWARTE1:
009C6C  0681 00000001    ADD.L #1,D1    * ABZAEHLEN  0-DAUER
009C72  0810 0000        BTST #BITNR,(A0)
009C76  6700 FFF4        BEQ TWARTE1
009C7A                   TWARTE2:
009C7A  0682 00000001    ADD.L #1,D2    * ABZAEHLEN 1-DAUER
009C80  0810 0000        BTST #BITNR,(A0)
009C84  6600 FFF4        BNE TWARTE2
009C88                   *
009C88  4E75             RTS
009C8A
009C8A                   FILL:          * MIT LEERZEICHEN FUELLEN
009C8A  3C3C 0014        MOVE #20,D6
009C8E                   LPA:
009C8E  10FC 0020        MOVE.B #' ',(A0)+
009C92  51CE FFFA        DBRA D6,LPA
009C96  4210             CLR.B (A0)
009C98  4E75             RTS
009C9A
009C9A
009C9A                   START:         * HIER HAUPTPOGRAMM
009C9A  6100 FF64        BSR FREQMESSE
009C9E  41F9 00009D6F    LEA BUFFER,A0
009CA4  2001             MOVE.L D1,D0
009CA6  4EB9 000E15FC    JSR @PRINT4D   * AUSGABE DES MESSWERTES
009CAC  6100 FFDC        BSR FILL
009CB0  41F9 00009D5F    LEA TEXT,A0
009CB6  323C 000A        MOVE #10,D1
009CBA  343C 00C8        MOVE #200,D2
009CBE  303C 0022        MOVE #$22,D0
009CC2  4EB9 000E1386    JSR @WRITE     * AUF DEN BIDSCHIRM
009CC8                   *
009CC8  6100 FF88        BSR TIMMESSE
009CCC  41F9 00009D44    LEA TEXT1,A0   * LOW ZU HIGH
009CD2  48E7 6000        MOVEM.L D1/D2,-(A7)
009CD6  323C 000A        MOVE #10,D1
009CDA  343C 0064        MOVE #100,D2
009CDE  303C 0022        MOVE #$22,D0
009CE2  4EB9 000E1386    JSR @WRITE     * AUF DEN BIDSCHIRM
009CE8  4CDF 0006        MOVEM.L (A7)+,D1/D2
009CEC                   *
009CEC  3601             MOVE D1,D3
009CEE  D642             ADD D2,D3                          zu Abb. 7.10.12
009CF0  C3FC 0064        MULS #100,D1
009CF4  83C3             DIVS D3,D1     * PROZENTERGEBNIS
009CF6  3001             MOVE D1,D0
009CF8  41F9 00009D6F    LEA BUFFER,A0
```

```
Rolf-D.Klein 68000/08 Assembler 4.3 (C) 1984, Seite 3

009CFE  4EB9 000E15FC     JSR @PRINT4D
009D04  6100 FF84         BSR FILL
009D08  41F9 00009D6F     LEA BUFFER,A0
009D0E  323C 001E         MOVE #30,D1
009D12  343C 0050         MOVE #80,D2
009D16  303C 0022         MOVE #$22,D0
009D1A  4EB9 000E1386     JSR @WRITE      * AUF DEN BILDSCHIRM
009D20  41F9 00009D5D     LEA TEXT2,A0
009D26  323C 0064         MOVE #100,D1
009D2A  343C 0050         MOVE #80,D2
009D2E  303C 0022         MOVE #$22,D0
009D32  4EB9 000E1386     JSR @WRITE
009D38                   *
009D38  4EB9 000E09D8     JSR @CSTS
009D3E  6700 FF5A         BEQ START
009D42  4E75              RTS
009D44
009D44
009D44  302D4461756572  TEXT1: DC.B '0-Dauer  zu  Gesamtdauer' ,0
009D4B  20207A75202047
009D52  6573616D746461
009D59  75657200
009D5D  2500            TEXT2: DC.B '%',0
009D5F  4672657175656E  TEXT: DC.B 'Frequenz (Hz) = '   * OHNE 0 ALS ABSCHLUSS
009D66  7A2028487A2920
009D6D  3D20
009D6F                  BUFFER: DS.B 80
009DBF
009DBF                  END

0E8C22  Ende-Symboltabelle
```

Abb. 7.10.12 Ein Programm zum Abgleich der CAS-Baugruppe

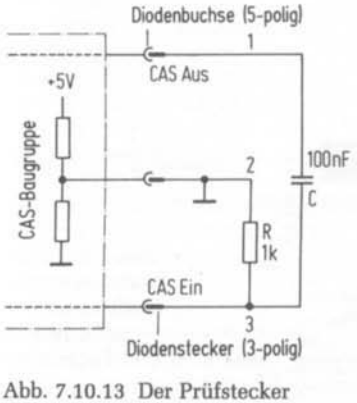


Abb. 7.10.13 Der Prüfstecker zum Abgleich der Baugruppe

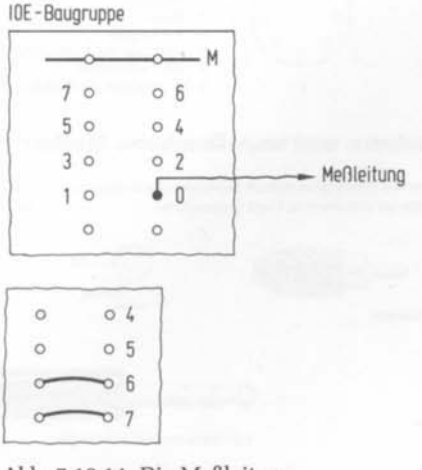
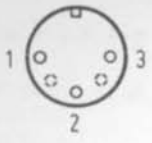


Abb. 7.10.14 Die Meßleitung auf der IOE-Baugruppe

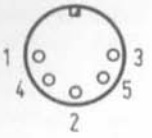
NF-Stecker und Buchsen



Mikrofonbuchse

Mono

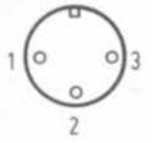
1 Aufnahme
2 Masse
3 Leer



Stereo L/R

1 Aufnahme, linker/rechter* Kanal
2 Masse
3 Leer
4 Aufnahme, rechter/linker* Kanal
5 Leer

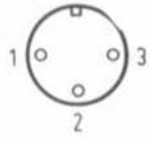
* Buchse für rechten Kanal bei getrennten Buchsen



TA-Buchse

alt

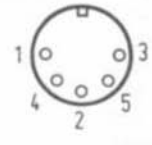
1 Rechter Kanal
2 Masse
3 Linker Kanal u. Mono



TA/TB-Buchse

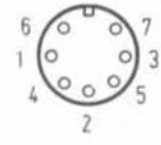
alt

1 Aufnahme TB
2 Masse
3 Wiedergabe TA/TB



TB-Buchse Stereo

1 Aufnahme Mono und Aufnahme Stereo, linker Kanal
2 Masse
3 Wiedergabe Mono und Stereo, linker Kanal
4 Aufnahme Stereo, rechter Kanal
5 Wiedergabe Stereo, rechter Kanal



Universalbuchse 7polig

1 Aufnahme
2 Masse
3 Wiedergabe
4 Leer oder mit 1 verbunden
5 Mit 3 verbunden
6, 7 Start/Stop-Fernbedienung mit Schaltmikrofon

Radiobuchse oder Tuner

Mono

1 Aufnahme
2 Masse
3 Wiedergabe

Stereo

1 Aufnahme linker Kanal
2 Masse
3 Wiedergabe, linker Kanal
4 Aufnahme, rechter Kanal
5 Wiedergabe, rechter Kanal

neu

1 Leer
2 Masse
3 Linker Kanal
4 Leer
5 Rechter Kanal

neu

1 Aufnahme TB
2 Masse
3 Wiedergabe TA/TB
4 Leer
5 Mit 3 verbunden, falls Stereo TA angeschlossen (beide Kanäle parallel)

Lautsprecherbuchse

Masseanschluß

(zusätzlicher Anschluß = Umschalter)

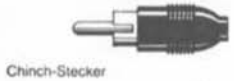
Stereo-Kopfhörer

Stecker

1 Leer
2 Linke Masse
3 Rechte Masse
4 Linkes Signa
5 Rechtes Sigr

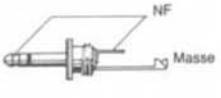
Japanische und amerikanische Stecker

Aufnahme und Wiedergabe müssen entweder separat gesteckt werden oder es wird intern im Gerät umgeschaltet.



Chinch-Stecker

6,3-mm-, 3,5-mm- und 2,5-mm-Klinkenstecker



6,3-mm-Stereo-Klinkenstecker

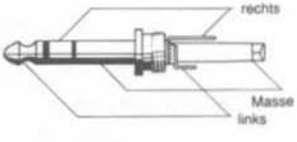


Abb. 7.10.15
Verschiedene Buchsen an Cassettenrecordern und Tonbandgeräten

Stecker für Spannungsversorgung

wieder erkennbar sein. Sehr starke Verzerrungen kann unsere Schaltung nicht mehr rückgängig machen.

Abb. 7.10.16 zeigt eine Schaltung, die in extremen Fällen eingesetzt werden kann. Sie wirkt sich auf die Aufnahme aus. Manche Rekorder verarbeiten die TTL-Signale mit den steilen Flanken bei der Aufzeichnung nicht. Die Schaltung filtert die Flanken weg und verzerrt das Signal in günstiger Weise für diese Rekordergruppe.

Wenn Sie Probleme haben, fragen Sie auch bei den Bausatzherstellern an. Sie halten auch eine Liste von geprüften Rekordern bereit.

7.11 Die Floppy-Baugruppe FLO2

Zum Anschluß eines Disketten-Laufwerks benötigt man eine spezielle Baugruppe: Die FLO2. Auf dieser Baugruppe werden alle notwendigen Anpassungen durchgeführt. Dabei können unterschiedliche Laufwerke (8", 5 1/4", 3 1/2" und 3") angeschlossen werden. Bis zu vier Laufwerke sind insgesamt anschließbar (mit binärer Codierung sogar 15).

Materialliste:

IC1,IC6 7405 Inverter mit offenem Kollektor
IC2 74LS14 Invertierende Schmitt-Trigger
IC3 9229B Datenseparator und Präkompensator
IC4 1797 Floppy-Disk-Controller
IC5 74LS367 Nicht-Invertierende Treiber
IC7 74LS38 Leistungs-Nand-Glieder
IC8 7404 Inverter
IC9 74LS32 Oder-Glieder
IC10,IC11 74LS245 bidirektionale Datenbustreiber
IC12 74LS273 Zwischenspeicher
IC13 74LS85 Vergleicher
IC14 74LS138 1-aus-8 Dekoder
1x 40polige IC-Fassung
3x 20polige IC-Fassung
3x 16polige IC-Fassung
6x 14polige IC-Fassung
1x Quarzoszillator 16 MHz
R1,R6,R7 10 kΩ 1/8 W (unkrit.)
R2,R3,R4,R5 3.3 kΩ
R8,R9,R10,R12,R13,R14 1 kΩ
R11 4.7 kΩ
AR1 4x 3.3 kΩ Widerstandsnetzwerk
AR2 8x 1 kΩ (oder früher 3.3k Ω)
Widerstandsnetzwerk
C1,C6,C7 10 µF 16V
C2,C3,C4,C5 100 nF
ST Stiftleiste, gewinkelt, 36polig
ST1 doppelreihige Stiftleiste, gerade, 2*6polig
ST2 doppelreihige Stiftleiste, gewinkelt, 2*25polig
ST3 5 Einzelstifte

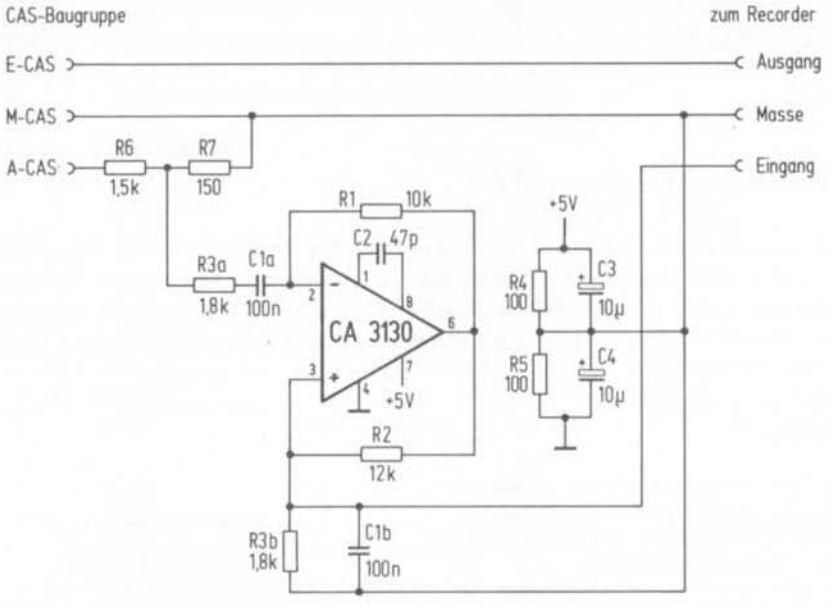


Abb. 7.10.16 Eine Schaltung für schwierige Fälle

ST4 doppelreihige Stiftleiste, gerade, 2*17polig
ST5 doppelreihige Stiftleiste, gerade, 2*4polig
1x Leiterplatte FLO2
1x Verbindungskabel Flachbandleitung, passend zum Disketten-Laufwerk
1x Spannungsversorgungs-Kabel für Diskettenlaufwerk
1x Diskettenlaufwerk (z.B. TEAC, 80-Spur, doppelseitig).

Kenndaten:

Spannungsversorgung: +5 V, Stromaufnahme 360 mA
Spannungsversorgung: +12 V, Stromaufnahme 20 mA
Die Laufwerke benötigen auch eine zusätzliche Spannungsversorgung. Je nach Laufwerk sind +5 V und +12 V bei den meisten Mini- und Mikrolaufwerken, oder +5 V und +24 V bei den meisten Maxi-Laufwerken erforderlich.
Dazu muß man im Datenblatt der Laufwerke nachsehen.

Abb 7.11.1 zeigt den kompletten Schaltplan der FLO2-Baugruppe. Über den bidirektionalen Bustreiber IC10 (74LS245) ist der Datenbus mit dem Controller verbunden. Ein Widerstandsnetz mit 1-kΩ-Widerständen sorgt dafür, daß keine Störungen auf der Versorgungsleitung auftreten, wenn der Buszustand am Controller von Tri-State auf Aktiv wechselt. Diese Störungen können vom 74LS245 neuer Generation verursacht werden, da der Baustein bei einem Wechsel des Eingangspegels am B-Eingang von offen nach aktiv plötzlich viel Strom zieht. Durch die Widerstände wird verhindert, daß am B-Eingang ein undefinierter Spannungspegel anliegt.

Am Vergleicher IC13 (74LS85) wird die Adresse der Baugruppe eingestellt. Sie wird auf die Adresse $FFFFFFC0 gelegt, was bedeutet, daß die Brücken gegenüber von A4 und A5 auf Masse geschaltet werden und die Brücken gegenüber von A7 und A6 offen bleiben. Auf dem Bestückungsplan ist die Lage der Brücken eingezeichnet.

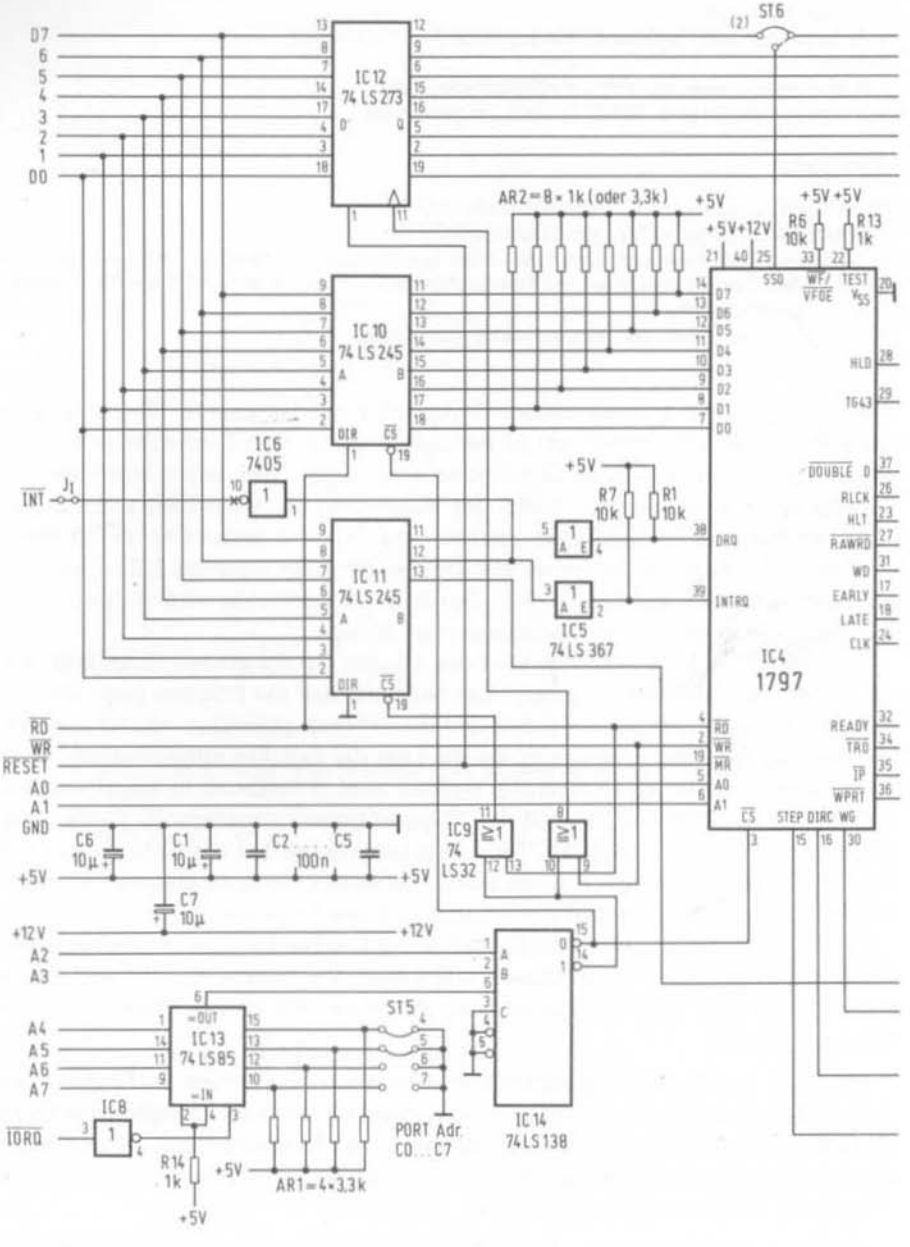
Von dem Dekoder IC14 (74LS138) werden zwei Adreßbereiche angesprochen. Von $FFFFFFC0 bis $FFFFFFC3 wird der Floppy-Controllerbaustein IC 4 selektiert, und zwischen $FFFFFFC4 und $FFFFFFC7 liegt beim Schreiben das IC 12, beim Lesen das IC 11. Dabei ist aber nur die Adresse 0C4H interessant, denn die anderen Adressen sind nicht eindeutig dekodiert und führen zum gleichen Ergebnis.

Das IC 13 (74LS273) dient dem Einstellen der Laufwerksnummer, des Laufwerkstyps (Mini oder Maxi), sowie der Schreib- und Lesedichte (single, double) und der Seitenauswahl. Optional kann man damit auch den Motor ein oder ausschalten, falls es das Laufwerk erlaubt.

Abb. 7.11.2 zeigt die Belegung des Ports. Die Bits 0 bis 3 dienen der Laufwerkseinstellung. Wenn ein Laufwerk angesprochen werden soll, wird das dazugehörige Bit gesetzt. Dabei kann die Laufwerkscodierung auch binär erfolgen, allerdings müssen dann auch die angeschlossenen Laufwerke diese Selektierung erlauben. Normalerweise ist nur ein Bit gesetzt, der Rest bleibt auf null.

Mit Bit 4 wird bestimmt, ob die Aufzeichnung in einfacher (FM) oder doppelter Dichte (MFM) erfolgt. Ist das Bit 4 auf gesetzt, wird die einfache Dichte verwendet.

Mit Bit 5 wird bestimmt, ob Mini- oder Maxi-Laufwerke verwendet werden. Dabei ist zu beachten, daß heute auch Mini- oder Mikrolaufwerke existieren, die man mit MAXI


D7
6
5
4
3
2
1
D0
IC 12
74 LS 273
ST6
(2)
AR2 = 8 × 1k (oder 3,3k)
+5V
+5V+12V
+5V +5V
R6
10k
R13
1k
IC 10
74 LS 245
IC6
7405
INT
J1
R7
10k
R1
10k
IC 11
74 LS 245
IC5
74 LS 367
IC4
1797
RD
WR
RESET
A0
A1
GND
C6
10µ
C1
10µ
C2 ... C5
100n
C7
10µ
+12V
IC9
74
LS32
A2
A3
A4
A5
A6
A7
IC13
74 LS 85
ST5
IC14
74 LS 138
IC8
IORQ
R14
1k
AR1 = 4 × 3,3k
PORT Adr.
C0 ... C7
DRQ
INTRQ
DOUBLE D
RLCK
HLT
RAWRD
WD
EARLY
LATE
CLK
READY
TRQ
IP
WPRT
STEP DIRC WG

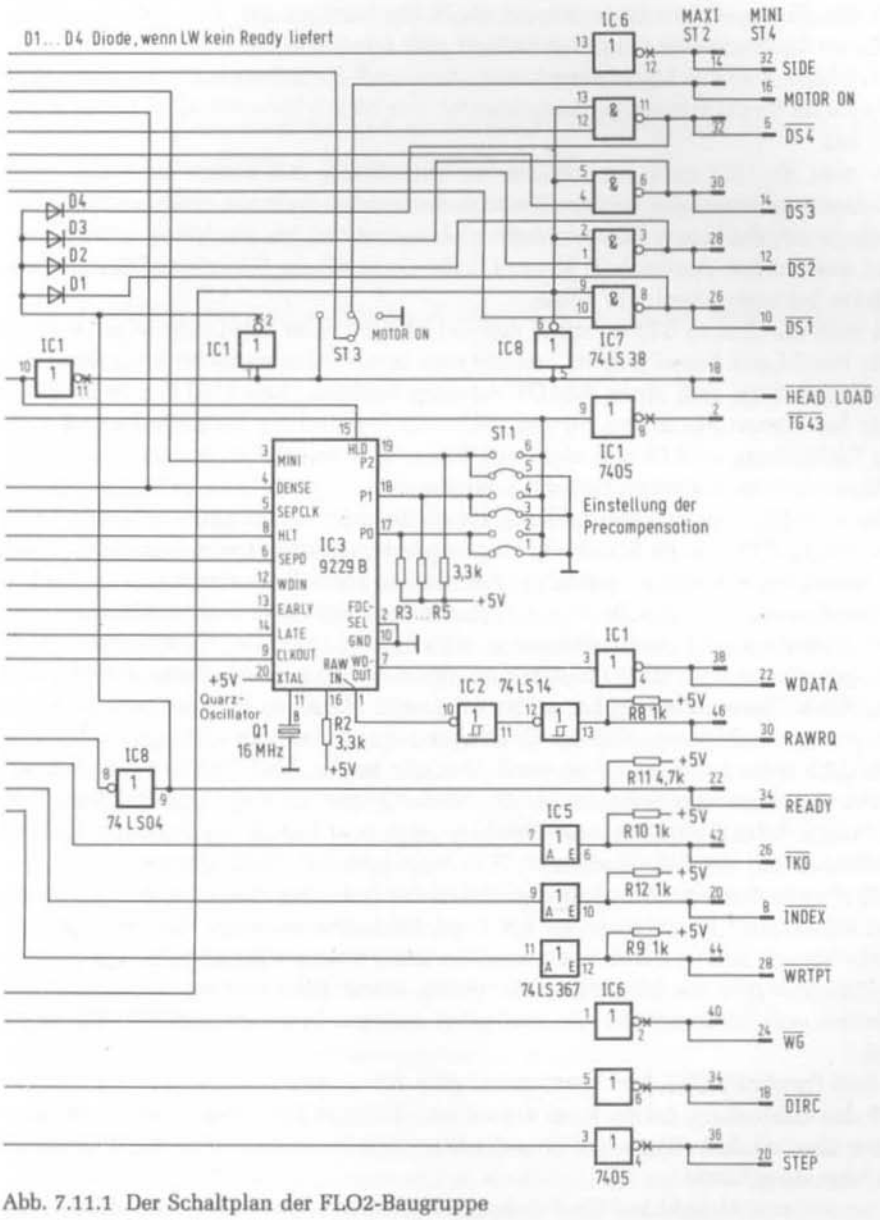


Abb. 7.11.1 Der Schaltplan der FLO2-Baugruppe

bearbeiten muß. Wird das Bit 5 auf 1 gesetzt, so erfolgt die MINI-Bearbeitung, und der Takt für den Floppy-Controller wird auf die Hälfte herabgesetzt. Die Aufzeichnungsdichte dieser Laufwerke ist dann nur halb so groß wie bei MAXI.

Mit Bit 6 kann man den Motor ausschalten; dazu muß die Brücke bei ST3 so eingebaut sein, wie sie im Bestückungsplan gezeichnet ist. Der Wert 0 bedeutet dabei Motor ein, 1 steht für aus.

Wenn man das Bit zum Ausschalten in Verbindung mit einem Laufwerk ohne READY-Ausgang verwendet, muß muß man darauf achten, daß man nach dem Einschalten durch den Aufruf einer Warteschleife so lange wartet, bis der Motor seine Nenndrehzahl erreicht hat. Andernfalls können insbesondere beim Schreiben Fehler auftreten; defekte Sektoren wären die Folge.

Wenn man die Brücke ST3 so stellt, daß der Motor immer dann eingeschaltet wird, wenn ein Head-Load-Signal auftritt, braucht man in der Software nichts zu ändern. Das Laufwerk muß dann aber einen READY-Ausgang besitzen (Anschluß Pin 34 an ST4). Achtung: bei älteren FLO2-Baugruppen fehlte die Verbindung nach Pin 34 von ST4, und die Verbindung zu ST3 war nicht von Pin 1, IC 1 hergeleitet, sondern von Pin 2. Sehen Sie sich bitte in diesem Fall das Layout an!

Mit Bit 7 wählt man die Floppy-Seite zweiseitig arbeitender Laufwerke aus. Dazu muß die Brücke ST6, wie im Schaltbild eingezeichnet, verdrahtet sein. Im Layout ist die Brücke bereits fest eingebaut; normalerweise ist also keine Veränderung an der Leiterplatte vornehmen. Wird das Bit 7 auf 1 gesetzt, so wird Seite 1, also die Rückseite, ausgewählt. Seite 0 wird verwendet, wenn man das Bit 7 auf den Wert 0 setzt.

Der Floppy-Controller FD1797 besitzt auch einen Ausgang für die Seitenauswahl, den Ausgang SSO. Dieser Ausgang hat leider folgenden Nachteil. Wird er verwendet, so kann man nicht verhindern, daß der Controller anhand des Seitenbits kontrolliert, ob auch wirklich Seite 1 angesprochen wird. Jetzt gibt es aber Disketten mit Programmen im Handel, bei denen das Seitenbit bei der Formatierung nicht auf 1 gesetzt wurde. In diesem Fall muß der Controller einen Befehl erhalten, auf Seite 0 zu arbeiten, obwohl er in Wirklichkeit auf die Seite 1 zugreift. Dies kann man mit der FLO2 tun.

Der FD1793 besitzt eine Abschaltmöglichkeit der Seitenumschaltung, warum wurde er nicht verwendet? Er ist nicht in der Lage, Disketten zu lesen, die bei einfacher Schreibdichte mit GAP = 0 formatiert wurden. Diese Formatierung findet man aber bei vielen IBM-Anlagen, sie ist daher interessant, wenn man Disketten im 8"-Format übernehmen will. Vielleicht ist der Fehler bei neueren Versionen des FD1793 schon beseitigt.

Nun zum Port $FFFFFFC4 im Lesebetrieb. *Abb. 7.11.3* zeigt die Belegung. Bit 7 ist das DRQ-Bit des Controllers. Leider kann man dieses Bit nicht (ohne Nachteile) direkt vom Controller übernehmen. Wenn das Bit auf 1 liegt, so will der Controller ein Byte holen, oder es liegt eines bereit.

DRQ wurde mit Absicht auf Bit 7 gelegt, um eine schnelle Abfrage des Bits durch einen ROL.B-Befehl zu ermöglichen. Das Bit wird damit nämlich in das Carry-Flag transferiert; das Ergebnis ist dann einfach durch einen Sprung „BCS" oder „BCC" auszuwerten.

Genauso verhält es sich mit dem Bit 6, dort liegt der INTRQ-Ausgang des Controllers. Das Bit wird auf 1 gesetzt, wenn ein Interrupt des Controllers vorliegt. Nun kann man

zum einen einen echten Interrupt auslösen, wie es beim Z80 getan wird, oder das Bit abfragen, wie es bei den 68000/68008-Routinen geschieht. Das Bit gelangt durch einen ROL.B-Befehl in das Vorzeichenbit. Mit BMI, BPL kann in Abhängigkeit vom Ergebnis gesprungen werden.

Schließlich gibt es noch das Bit 5 mit der Headload-Information. Wenn der Kopf noch auf der Diskette aufliegt, so ist das Bit auf 1 gesetzt.

Damit kann man vor einem Schreib- oder Lesebefehl abfragen, ob der Kopf noch auf der Diskette aufliegt; wenn ja, braucht man ihn nicht erneut zu laden und spart sich 15 ms Kopfladezeit.

Die restlichen Bits sind unbelegt und können später einmal Erweiterungen dienen.

Der Baustein 9229:

Dabei handelt es sich um einen universellen Baustein, der den Aufbau von Floppy-Controller-Schaltungen stark vereinfacht. Das IC beinhaltet neben dem Datenseparator auch eine Präcompensations-Schaltung.

Der Datenseparator besteht aus einer sehr ausgeklügelten Schaltung, die alle Vorteile einer früher analog realisierten Schaltung mit denen der Digitaltechnik verbindet. So ist z. B. kein Abgleich erforderlich, was den Nachbau der FLO2 erheblich vereinfacht.

Der Präcompensator besitzt drei Eingänge P0, P1 und P2, mit denen man die Präcompensationszeit einstellen kann. In der Schaltung wird dies mit den Brücken bei ST1 getan. Dabei ist wichtig, daß manche Floppy-Laufwerke erst eine Präcompensation ab Spur 43 benötigen. Aus diesem Grund existieren drei Beschaltungs-Möglichkeiten für jeden der Eingänge P0 bis P2. Einmal bleibt der Eingang offen, bzw. liegt über einen Widerstand an +5 V. In diesem Fall ist die Präcompensation dauerhaft eingestellt. Als zweite Einstellung wird der Eingang auf 0 V geschaltet; so ist das betreffende Bit unwirksam. Schließlich kann der entsprechende Eingang auf den Ausgang TG43 des Floppy-Controllers gelegt werden, dann ist die Präcompensation erst wirksam, wenn eine Spur größer gleich 43 angefahren wird.

Für normale, moderne Laufwerke genügt es, alle Eingänge auf 0 zu schalten, also keine Präcompensation zu verwenden. In den technischen Handbüchern der Laufwerke steht ein Hinweis, falls eine Präcompensation erforderlich ist.

Abb. 7.11.4 zeigt die Tabelle der möglichen Präcompensationszeiten. *Abb. 7.11.5* zeigt die Einstellung für ein MAXI-Laufwerk mit mehr als 40 Spuren, das eine Präcompensation von 62,5 ns benötigt, wenn eine Spur größer gleich 43 bearbeitet wird.

Man kann auch Präcompensationszeiten einstellen, die auf allen Spuren gleich sind, oder ab einer bestimmten Spur einen höheren Wert annehmen. Wie schon gesagt, die meisten Laufwerke benötigen keine Präcompensation, das gilt insbesondere für Minilaufwerke mit 40 Spuren.

Der Vollständigkeit halber sei nochmals darauf hingewiesen, daß die Präcompensation nur bei doppelter Aufzeichnungsdichte relevant ist; bei einfacher Dichte wird keine verwendet.

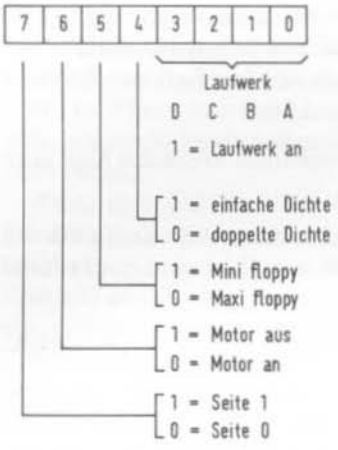


Abb. 7.11.2 Die Belegung des Ports $FFFFFFC4 beim Schreiben

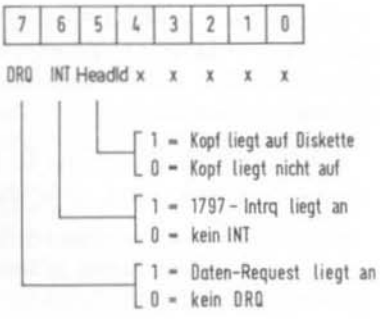


Abb. 7.11.3 Die Belegung des Ports $FFFFFFC4 beim Lesen

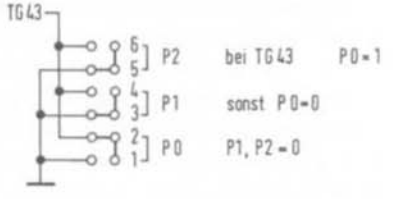


Abb. 7.11.5 Ein Einstellbeispiel

Mini = 0 (also Maxi)			
P2	P1	P0	
0	0	0	0 ns
0	0	1	62,5 ns
0	1	0	125 ns
0	1	1	187,5 ns
1	0	0	250 ns
1	0	1	250 ns
1	1	0	312,5 ns
1	1	1	312,5 ns
Mini = 1 (also Mini)			
0	0	0	0 ns
0	0	1	125 ns
0	1	0	150 ns
0	1	1	375 ns
1	0	0	500 ns
1	0	1	500 ns
1	1	0	625 ns
1	1	1	625 ns

Abb. 7.11.4 Verschiedene Präkompensations-Zeiten

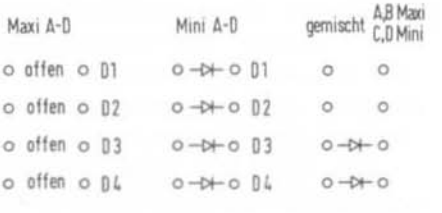


Abb. 7.11.6 Die Einstellung der Dioden

Bei der Präcompensation werden nahe beinander stehende Datenbits noch näher zusammengedrängt, da sie bei der Wiedergabe dazu neigen, von sich aus weiter entfernt zu erscheinen. Die Präcompensationszeit gibt an, um wieviel sie zusammengedrängt werden. Wählt man die Zeit zu groß, so gibt es Datenfehler.

Für die ersten Versuche setzen wir also die Brücken 1, 3 und 5 bei ST1 ein.

Die Steuerleitungen:

Eine andere wichtige Funktion erfüllt das READY-Signal. Bei allen MAXI-Laufwerken ist es meist, aber nicht immer, vorhanden und liegt auf Pin 22 der 50poligen Steckerleiste. Bei Minilaufwerken ist es meist nicht vorhanden. Achtung: Bitte unbedingt im Handbuch zum jeweiligen Laufwerk nachsehen! Praktisch alle modernen Laufwerke geben ein solches Signal aus, während es bei älteren Laufwerken nicht vorhanden ist.

Wenn es bei einem MAXI-Laufwerk fehlt, so muß man auch eine Diode einsetzen. Ist ein READY-Ausgang vorhanden, liegt er meist auf dem Stift 34 der 34poligen Steckerleiste und kann per Hand auf den gemeinsamen Ready-Eingang der Dioden (Anode) D1 bis D4 verdrahtet werden. Die Dioden werden dann nicht bestückt.

Die Dioden sind nur in Verbindung mit Laufwerken ohne READY-Signal relevant. Werden Drives mit und ohne READY-Signal gemischt eingebaut, sind die Dioden nur für solche Laufwerke erforderlich, die keinen READY-Ausgang besitzen. *Abb. 7.11.6* zeigt Beispiele für die Diodencodierung.

Nachdem uns die weiteren Signale nur am Rande interessieren, hier eine kurze Beschreibung:

Mit SIDE wird die Laufwerksseite bestimmt. SIDE liegt auf 0, wenn die Seite 1 ausgewählt wird, bei Seite 0 liegt es auf 1.

MOTOR ON wird meist nur für Minilaufwerke verwendet. Es liegt auf 0, wenn der Motor an sein soll.

DS1-Quer bis DS4-Quer liegt auf 0, wenn das Laufwerk angesprochen wird. Achtung: Bei manchen Minilaufwerken gibt es kein DS4-Quer!

Mit HEAD-LOAD-QUER wird der Laufwerkskopf geladen. Es ist bei uns mit dem Selekt-Signal gekoppelt.

TG43-Quer liegt auf 0, wenn eine Spur größer gleich 43 angesprochen wird, und wird nur von älteren MAXI-Laufwerken verwendet, um den Schreibstrom zu reduzieren. Manchmal liegt der Anschluß auch am Stift 8 und nicht an 2.

WDATA sind die Schreibdaten. RAWRQ sind die Lesedaten. TK0-Quer liegt auf 0, wenn der Kopf auf Spur 0 liegt. Es gibt ein paar Laufwerke, bei denen die Positionierung auf Spur 0 nicht funktioniert. Dort kann man auf Spur –1 fahren. Meist läßt sich das durch eine mechanische Neueinstellung beheben.

WRTPRT-Quer ist der Schreibschutzausgang. Damit kann das Laufwerk signalisieren, daß ein Schreibschutz gesetzt ist.

WG-Quer ist ein Freigabesignal für den Schreibausgang. DIRC-Quer und STEP-Quer dienen der Ansteuerung des Schrittmotors zur Spureinstellung.

Der Floppy-Controller:

Der Floppy-Controller wird über die Adressen $FFFFFFC0 bis $FFFFFFC3 angesprochen. *Tab. 7.11.1* zeigt die Belegung. Die Adresse $FFFFFFC0 spricht beim Schreiben das Befehlsregister an. Man kann dort Befehle an den Controller übergeben. Beim Lesen erfährt man den Status; er gibt z. B. darüber Auskunft, ob ein Lesefehler aufgetreten ist oder nicht.

Das Register $FFFFFFC1 ist das Spur-Register, dort steht die aktuelle Spurnummer, die von 0 bis maximal 255 reichen darf. Bei Maxilaufwerken ist die größte Zahl 76, und bei Minilaufwerken dürfen die Werte 34, 39 oder 79 nicht überschritten werden.

$FFFFFFC2 ist das Sektorregister, dort steht die aktuelle Sektornummer. Im Register $FFFFFFC3 werden Parameter und Daten vom oder zum Controller übergeben.

$FFFFFFC4 gehört, wie schon gesagt, nicht zum Floppy-Controller.

Nun zu den Befehlen. *Tab. 7.11.2* zeigt eine Liste davon. Man unterscheidet vier Typen. Die erste Gruppe (I) sind Befehle für die Kopfpositionierung, die Gruppe II ist zum Lesen und Schreiben von Sektoren vorgesehen, die Gruppe III dient der Kontrolle oder zum Formatieren, und in der Gruppe IV besteht aus einem Befehl für die Interruptsteuerung.

Generell aktiviert der Controller immer nach der Ausführung eines Befehls die Leitung INT. Damit wird das Bit 6 am Port 0C4h gesetzt. Der Prozessor wird immer dann die Interrupt-Bearbeitung aufnehmen, wenn er dazu durch den entsprechenden Befehl freigegeben war (Register SR beim 68008). Holt sich der Prozessor dann den Inhalt des Status-Registers ab, wird der Interrupt wieder gelöscht, und Bit 6 wird auf 0 zurückgesetzt.

Bei den Befehlen sind einige Optionen möglich. So kann man die Steprate bestimmen, mit der die Kopfpositionierung durchgeführt wird. *Tab. 7.11.3* zeigt die Umrechnungstabelle. Dann gibt es noch einige Hilfsbits, deren Funktionen in der *Tab. 7.11.4* aufgelistet sind. Wenn man das Bit mit der Bezeichnung „h" auf 1 setzt, so wird der Kopf zu Beginn des Befehls geladen, sonst erfolgt nur ein Update der internen Register. Bei den STEP-Befehlen muß zum update zusätzlich das Bit u auf 1 gesetzt sein.

Wird das Bit v auf 1 gesetzt, so erfolgt ein Prüfvorgang. Der Kopf eines Records wird angelesen und der Inhalt mit der aktuellen Spurnummer und ggf. Seitennummer verglichen. Stimmt der Wert nicht überein, folgt eine Fehlermeldung.

Beim Befehl „Spur suchen" wird die einzustellende Spur im Port-Register $FFFFFFC3 übergeben. STEP schreitet einen Schritt in die Richtung, in die zuletzt geschritten wurde. STEP-IN schreitet einen Schritt in Richtung Spur 76 (80 etc.). STEP-Out schreitet einen Schritt in Richtung Spur 0.

Beim Schreiben oder Lesen wird zusätzlich die Sektorinformation im Register $FFFFFFC2 ausgewertet. Dieser Sektor wird angewählt. Normalerweise fängt man bei Sektoren mit 1 an zu zählen, hingegen bei Spuren mit 0.

Die Daten werden im Register $FFFFFFC3 ausgetauscht. Beim Schreiben kündigt das DRQ-Signal die Anforderung eines neuen Wertes an, beim Lesen zeigt DRQ an, daß ein Byte vorliegt. Das DRQ-Bit wird jeweils gelöscht, wenn man die Anforderung erfüllt.

Wird einer solchen Aufforderung nicht Folge geleistet, übergibt der Controller im Statusregister eine Fehlermeldung.

	Lesen	Schreiben	
C0	Status	Befehl	
C1	Spur-Register	Spur-Register	
C2	Sektor-Register	Sektor-Register	1797
C3	Daten-Register	Daten-Register	
C4	Zusatz-Info	Auswahl	Zusatz

Tab. 7.11.1
Die Register des FD1797

Typ	Befehl	Befehl	7	6	5	4	3	2	1	0
I	Restore	Auf Spur 0	0	0	0	0	h	V	R0	R1
I	Seek	Spur suchen	0	0	0	1	h	V	R0	R1
I	Step	Schreiten	0	0	1	u	h	V	R0	R1
I	Step In	Schreite nach innen	0	1	0	u	h	V	R0	R1
I	Step Out	Schreite nach außen	0	1	1	u	h	V	R0	R1
II	Read Sektor	Lies Sektor	1	0	0	m	F_2	E	F_1	0
II	Write Sektor	Schreibe Sektor	1	0	1	m	F_2	E	F_1	a_0
III	Read Address	Lies Adresse	1	1	0	0	0	E	F_1	0
III	Read Track	Lies Spur	1	1	1	0	0	E	F_1	0
III	Write Track	Schreibe Spur	1	1	1	1	0	E	F_1	0
IV	Force Interrupt	Interrupt auslösen	1	1	0	1	I_3	I_2	I_1	I_0

Tab. 7.11.2 Die Befehle des FD1797

R1	R0	Maxi	Mini	Monitor
0	0	3 ms	6 ms	0
0	1	6 ms	12 ms	1
1	0	10 ms	20 ms	2
1	1	15 ms	30 ms	3

Tab. 7.11.3
Verschiedene Stepraten

h = 1	Kopf bei Start laden
h = 0	Kopf bei Start heben
v = 1	Spur prüfen durch Anlesen
v = 0	keine Prüfung
u = 1	Spur-Register auf Stand bringen
u = 0	Spur-Register belassen
m = 0	einen Record bearbeiten
m = 1	mehrere Records bearbeiten
$d_0 = 0$	Data-Mark FB
$d_0 = 1$	Deleted Data-Mark F8
$F_2 = 1$	Sektorlänge = Std. (IBM) 128, 256, 512, 1024
$F_2 = 0$	spezielle Längen = 256, 512, 1024, 128
$F_1 = 0$	SSO auf 0 (Seite 0 prüfen)
$F_1 = 1$	SSO auf 1 (Seite 1 prüfen)
$I_0 = 1$	Nicht Ready → Ready gibt INT
$I_1 = 1$	Ready → Nicht Ready gibt INT
$I_2 = 1$	Index Puls
$I_3 = 1$	sofort INT auslösen
$I_3 - I_0 = 0$	Stop ohne Interrupt

Tab. 7.11.4 Die Bedeutung von einzelnen Bits der Befehle

	7	6	5	4	3	2	1	0
Befehlstyp I	1 = Nicht Ready	1 = Schreibschutz	1 = Kopf geladen	1 = Suchfehler	1 = CRC-Fehler	1 = Spur 0	1 = Index	1 = Busy
Befehlstyp II, III	1 = Nicht Ready Der Befehl wird nicht ausgef. bis Ready 0	1 = Schreibschutz	1 = Typ-Fehler 1 = Deleted Data Mark oder Schreibfehler	1 = Record nicht gefunden	1 = CRC-Fehler	1 = Datenverlust	1 = DRQ	1 = Busy

Tab. 7.11.5 Das Status-Register

	Einfache Dichte	Doppelte Dichte
00–F4	Daten FM	Daten MFM
F5	–	A1* MFM, CRC löschen
F6	–	C2** MFM
F7	2 CRC-Bytes	2 CRC-Bytes
F8–FB	F8–FB mit CLK = C7, CRC löschen	F8–FB MFM
FC	FC mit CLK = D7	FC MFM
FD	FD mit CLK = FF	FD MFM
FE	FE mit CLK = C7, CRC löschen	FE MFM
FF	FF mit CLK = FF	FF MFM

* 4,5-Takt-Bit auslassen; ** 3,4-Takt-Bit auslassen

Tab. 7.11.6 Reservierte Datenwerte beim Formatieren

Die Belegung des Statusregisters zeigt *Tab. 7.11.5*. Sie ist abhängig vom ausgeführten Befehl und unterscheidet Typ I und Typ II/III-Befehle.

Mit dem Befehl „Read Adress" kann man den Startkopf eines Sektors lesen, der Auskunft über aktuelle Spur, Seite, Sektor und Sektorenlänge gibt. Die Informationen werden in der angegebenen Reihenfolge nacheinander im Register $FFFFFFC3, also dem Datenregister, mit einer DRQ-Anforderung übergeben.

Mit dem Befehl „Lies Spur" wird eine komplette Spur, beginnend beim Index-Loch bis zum erneuten Auftreten des Indexes eingelesen. Dabei werden nicht nur die Daten übertragen, sondern auch als Zusatzinformationen und Datenlücken. Die Daten sind immer korrekt, da sich der Controller bei Synchrontakten immer wieder auf den neuen Datenstrom einstellt. Dieser Befehl ist aber weniger zum Lesen, als zu Kontrollzwecken gedacht.

Der Befehl „Write Track" schließlich dient dem Formatieren einer Diskette. Es werden bestimmte Bytes als Steuerinformation zur Ablage von CRC oder Synchronisationsbits interpretiert. *Tab. 7.11.6* zeigt die Bedeutung des Bytes.

Abb. 7.11.7 zeigt das Beispiel die Formatierung nach IBM mit einfacher Dichte auf 8". *Abb. 7.11.8* ein Beispiel für die Formatierung mit doppelter Dichte auch auf 8".

Nun bleibt noch der Befehl „Force Interrupt". Damit kann man zum einen den Controller rücksetzen, zum anderen das Interruptverhalten einstellen.

Aufbau und Test:

Abb. 7.11.9 zeigt die Lötseite, *Abb. 7.11.10* die Bestückungsseite der Baugruppe. In *Abb. 7.11.11* ist der Bestückungsplan abgedruckt.

Aufbau:

1. Beim Aufbau beginnt man mit der Bestückung der IC-Fassungen. Dann werden alle passiven Bauteile, wie Widerstände und Kondensatoren, eingelötet, sowie die Stiftleisten. Als nächstes lötet man den Quarzoszillator Q1 ein. Dabei ist auf die richtige Orientierung zu achten. Der Quarzoszillator besitzt an einer Gehäuseseite einen Punkt. Dieser Punkt ist der Pin 1 und muß mit dem auf dem Bestückungsplan übereinstimmen. Beim Quarzoszillator darf man nicht zu lange löten. Für ihn wäre auch eine Fassung erhältlich. Da er aber nur vier Anschlüsse besitzt, wäre die mechanische Stabilität nicht sehr hoch.
2. Einschalten und auf Kurzschluß prüfen. An Pin 40 des IC 4 müssen +12 V liegen und an Pin 21 +5 V.

   Wenn man die ROA64 und CPU68K mit GDP und KEY dazusteckt (vorher ausschalten), so muß sich das Grundprogramm melden.
3. Ausschalten und alle ICs außer dem Diskettencontroller IC4 einsetzen.
4. *Abb. 7.11.12* zeigt die Belegung der Brücken für den ersten Versuch. Löten Sie auch zunächst alle Dioden ein!

NUMBER OF BYTES	HEX VALUE OF BYTE WRITTEN
40	FF (or 00)[1]
6	00
1	FC (Index Mark)
26	FF (or 00)
6	00
1	FE (ID Address Mark)
1	Track Number
1	Side Number (00 or 01)
1	Sector Number (1 thru 1A)
1	00
1	F7 (2 CRC's written)
11	FF (or 00)
6	00
1	FB (Data Address Mark)
128	Data (IBM uses E5)
1	F7 (2 CRC's written)
27	FF (or 00)
247**	FF (or 00)

*Write bracketed field 26 times

**Continue writing until FD179X interrupts out. Approx. 247 bytes.
1-Optional '00' on 1795/7 only.

Abb. 7.11.7
Die Formatierung von 8"-FM

NUMBER OF BYTES	HEX VALUE OF BYTE WRITTEN
80	4E
12	00
3	F6
1	FC (Index Mark)
50*	4E
12	00
3	F5
1	FE (ID Address Mark)
1	Track Number (0 thru 4C)
1	Side Number (0 or 1)
1	Sector Number (1 thru 1A)
1	01
1	F7 (2 CRCs written)
22	4E
12	00
3	F5
1	FB (Data Address Mark)
256	DATA
1	F7 (2 CRCs written)
54	4E
598**	4E

* Write bracketed field 26 times
**Continue writing until FD179X interrupts out. Approx. 598 bytes.

Abb. 7.11.8
Die Formatierung von 8"-MFM

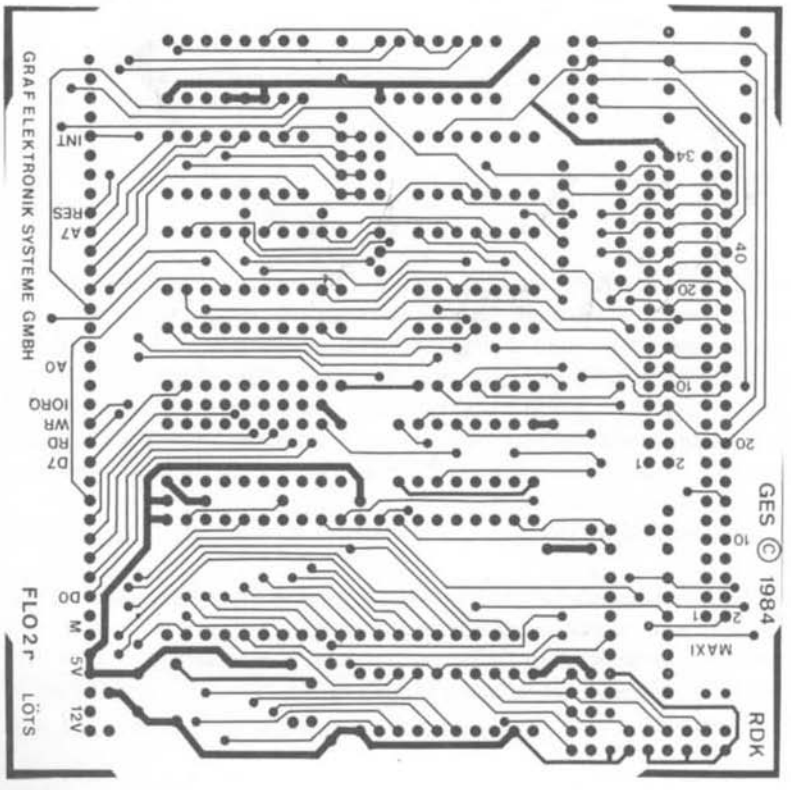


Abb. 7.11.9
Die Lötseite der Leiterplatte FLO2

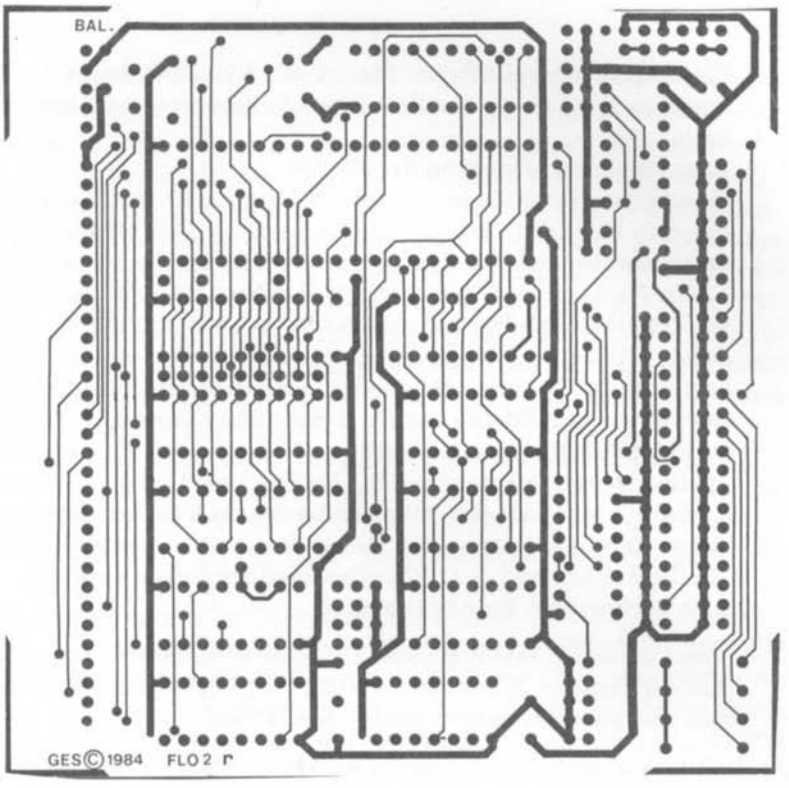


Abb. 7.11.10
Die Bestückungsseite der Leiterplatte FLO2

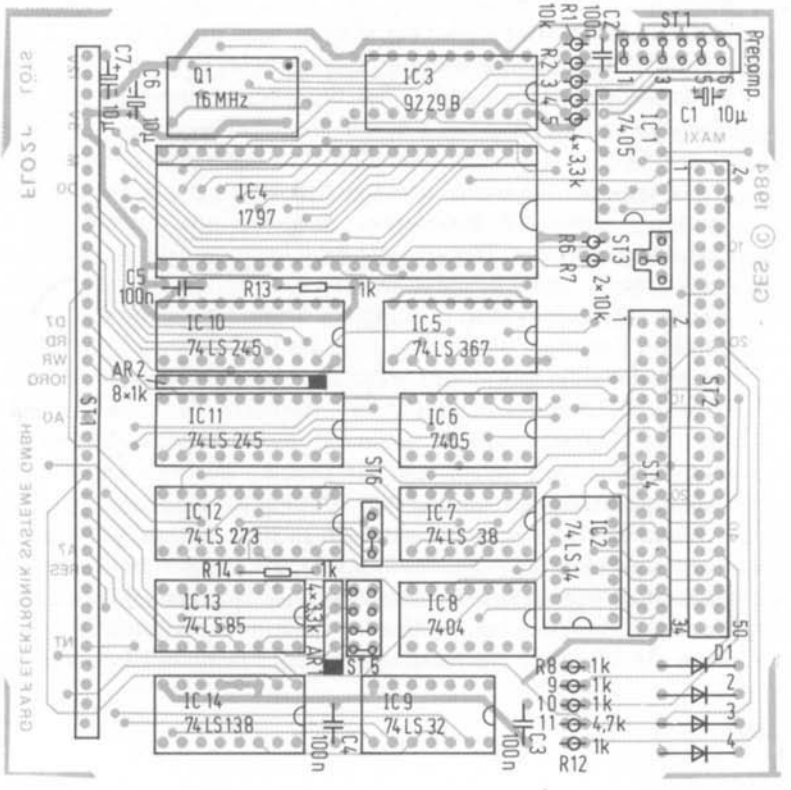


Abb. 7.11.11
Bestückungsplan der Baugruppe FLO2

5. Einschalten; messen Sie an Pin 24 des IC4. Dort muß ein Takt anliegen (die Frequenz ist abhängig vom Inhalt des Zwischenspeichers und noch nicht interessant).
6. Ausschalten und Einsetzen des FD1797.
7. Einschalten. Nun gibt man den Wert $55 auf den IO-Port $FFFFFFC1 aus. Dies kann mit dem Befehl „IO-Setzen" erledigt werden. Wird hinterher mit „IO-Lesen" der Inhalt des Ports wieder abgefragt, so muß $55 wieder erscheinen.

   Nun führt man den Test noch mit dem Wert $AA aus und hat damit geprüft, ob alle Datenleitungen ok sind, und ob die Adressierung stimmt.

Damit wurden der Datenbus und der FD1797 grob geprüft. Jetzt geht es an das Laufwerk. Das Laufwerk (Mini oder Maxi) wird über eine Flachbandleitung an die entsprechende Stiftleiste der FLO2-Baugruppe angesteckt. Dazu sollte man zuvor noch die Anleitung des Laufwerks studieren, um alle Einstellungen beim Laufwerk richtig durchgeführt zu haben. Das Laufwerk muß auf die Laufwerksadresse 1, DS1, A, also die erste Adresse, eingestellt sein (siehe auch Kapitel 6). Es benötigt eine eigene Versorgungsspannung. Neben den +5 V bedeutet dies normalerweise bei Minilaufwerken noch +12 V und bei Maxilaufwerken noch +24 V; manchmal sind auch noch weitere Spannungen nötig.

Jetzt kann der Test beginnen. Zunächst wird, falls vorhanden, eine formatierte Diskette eingelegt. Wenn keine formatierte Diskette greifbar ist, kann man den Test auch mal mit einer unformatierten durchführen.

Zum Port $FFFFFFC4 wird der Wert $11 bei Maxilaufwerken mit einfacher Dichte ausgeben, bei Minilaufwerken der Wert $21 für doppelte Dichte oder $31 für einfache Dichte. Dann wird der Wert $F zum Port $FFFFFFC0 gesandt. Damit wird ein RESTORE-Befehl ausgeführt. Das Laufwerk muß nun ansprechen; falls eine LED im Laufwerksgehäuse vorhanden ist, leuchtet diese für eine bestimmte Zeit auf. Nachdem nun der RESTORE-Befehl abgearbeitet wurde, lesen wir das Ergebnis vom Port $FFFFFFC0. Zur Interpretation etwa vorhandener Fehler dient Tab. 7.11.5.

Als Beispiel für einen erfolgreichen Versuch könnte der Binärwert 01100100 gelesen werden. Bit 6 bedeutet „Schreibschutz", wenn der Schreibschutz der Diskette gesetzt war. Das gesetzte Bit 5 bedeutet „der Kopf liegt noch auf".

Bit 2 weist darauf hin, daß sich der Kopf in der Spur 0 befindet. Er wurde durch den RESTORE-Befehl so positioniert. Wenn man etwas später abfragt, so erscheint der Wert 0 an allen Bit-Stellen, denn das Laufwerk wird nur für eine bestimmte Zeit selektiert. Fehler gibt es erst, wenn andere Bits auftauchen. So z.B. Bit 7. Das würde bedeuten: „Laufwerk nicht READY"; vielleicht steckt nur die Diskette falsch herum. Bit 0 kann gesetzt bleiben, wenn z.B. der Indexpuls nicht kommt. Man kann das einmal probieren, indem man die Diskette aus dem Laufwerk herausnimmt und dann den Befehl $0F an den Port $FFFFFFC0 ausgibt.

Wenn das Bit 4 oder 3 gesetzt ist, so liegt ein Lesefehler vor. Entweder stimmt dann die eingestellte Dichte nicht, oder die Diskette ist nicht formatiert. Außerdem kann auch ein Fehler in der Schaltung vorliegen. Man sollte dann neben der Kontrolle der einzelnen Lötstellen auch die Diskette überprüfen. Aufschluß gibt dann ein Blick auf ein Skop, wenn man die Diskettensignale RAWRD, RCLK am

Controller-IC und RAWRD-Quer am Controller-IC (IC 4) überprüft. Auch könnte der Takt fehlen (XTAL Pin 11 IC3, oder CLK Pin 24 IC 4).

Die *Abb. 7.11.13* bis *Abb. 7.11.19* zeigen Timing-Diagramme als Hilfestellung für eine Fehlersuche. Verschiedene Lese- und Schreibvorgänge wurden aufgenommen.

8. Nun ein weiterer Test: Der Such-Befehl. Damit wird der Schrittmotor kontrolliert. Auf den Port $FFFFFFC4 gibt man wieder den Laufwerkscode (Maxi SD = $11, Mini DD = $21), auf den Port $FFFFFFC3 gibt man die Spurnummer, z. B. $20, und danach auf den Port $FFFFFFC0 den Wert $1F, was dem SEEK-Befehl entspricht, aus. Dann muß das Laufwerk angesprochen und der Kopf zur angegebenen Spur gebracht werden. Anschließend kann man wieder den Status am Port $FFFFFFC0 kontrollieren; dort muß nach einer Weile der Wert 0, oder wenn man schnell genug mit der Abfrage ist, der Wert $20 oder $60.
   Im 68000-System müssen, wie bereits gesagt, alle Port-Adressen mit zwei multipliziert werden. In diesem System kommen nur gerade Adreßwerte vor. Für den eben beschriebenen Test bedeutet dies: $FFFFFFC0·2 bis $FFFFFFC4·2.
   Nachdem mit diesem Test auch das Lesen geprüft wurde, ist der größte Teil des Gesamttests beendet.
9. Jetzt wird eine Diskette formatiert. Dies ist sowohl als Test, als auch zur Bereitstellung einer gebrauchsvertigen Diskette gedacht. Das Programm UFORM68K prüft alle möglichen Fehlerquellen und gibt Hinweise auf dem Bildschirm (siehe Kapitel 6).
10. Wenn man eine automatische Motorabschaltung verwenden will, so stellt man die Brücke ST 3 wie in *Abb. 7.11.20* gezeigt ein. Kontrollieren Sie auch, ob das Laufwerk einen READY-Ausgang besitzt. Die entsprechende Diode D1 bis D4 muß dann entfernt werden (oder alle).

Die FLO2-Baugruppe muß dem neuen Stand entsprechen oder umgerüstet werden. Dazu ist eine Verbindung vom Pin 34, ST 4 zum READY-Signal (z.B. Pin 22 ST 2) herzustellen und außerdem eine Korrektur am IC1 Pin 1 nach ST3 erforderlich.

Abb. 7.11.21 zeigt das Timing beim Start des Laufwerks. Der Controller schaltet dann durch das Signal HLD das Laufwerk an. Das Signal HLT wird vom IC 9229 automatisch nach 80 ms geliefert. Der Controller greift erst dann zu, wenn auch das READY-Signal auf 1 geht. Somit ist sichergestellt, daß der Motor seine Nenndrehzahl erreicht hat.

Kontrollieren Sie in jedem Fall das READY-Signal nach. Falls es dauernd auf 1 liegt, greift der Controller sofort nach Ablauf von HLT zu. Dabei kann es, insbesondere beim Schreiben von Daten, zu Fehlern kommen, die man nur selten rechtzeitig bemerkt.

7.12 Die Baugruppe PROMMER

Wenn sofort nach dem Einschalten nicht nur das Grundprogramm, sondern auch eigene Programme zur Verfügung stehen sollen, kann man sie zu diesem Zweck in einem EPROM unterbringen. Daten können dabei nicht wie bei einem RAM vom System eingeschrieben werden; es müssen weitere Voraussetzungen gegeben sein. Dazu ist eine spezielle Schaltung nötig, nämlich ein EPROM-Programmiergerät. Die Baugruppe PROMMER ist in der Lage, EPROMs zu programmieren.

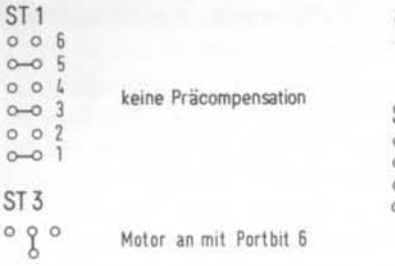


Abb. 7.11.12 Einstellung der Brücken für den ersten Test

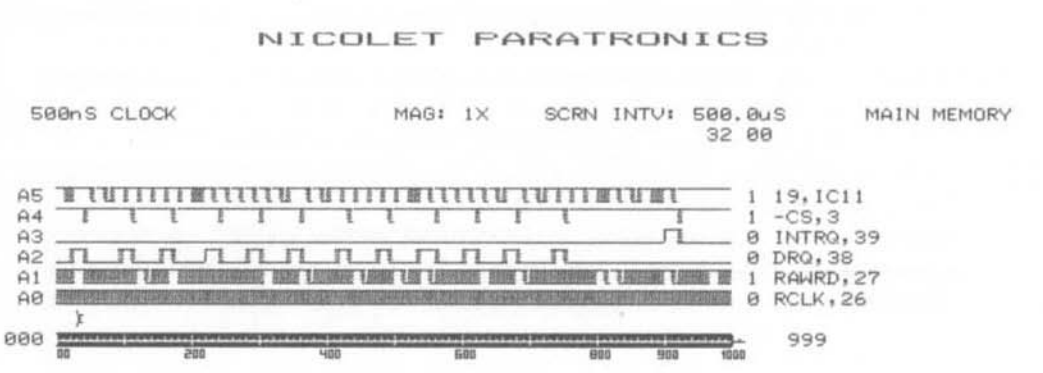


Abb. 7.11.13 Timing beim Lesen von Daten (500 µs)

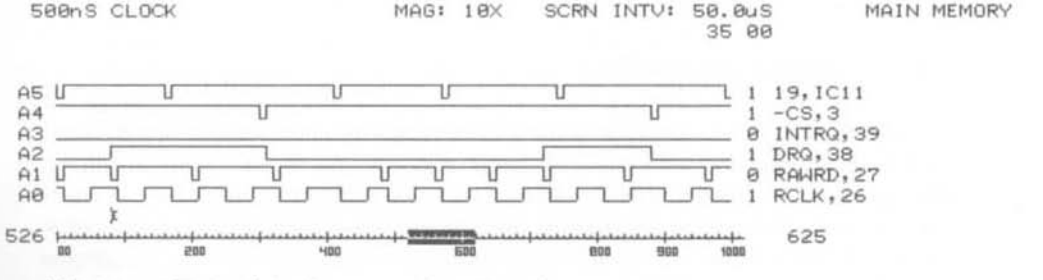


Abb. 7.11.14 Timing beim Lesen von Daten (50 µs)

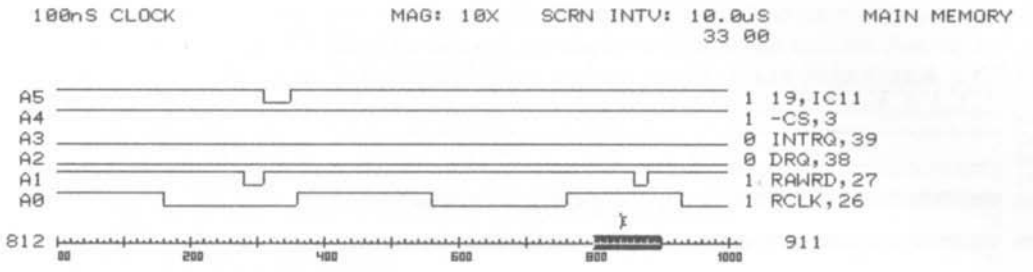


Abb. 7.11.15 Timing beim Lesen von Daten (10 µs)

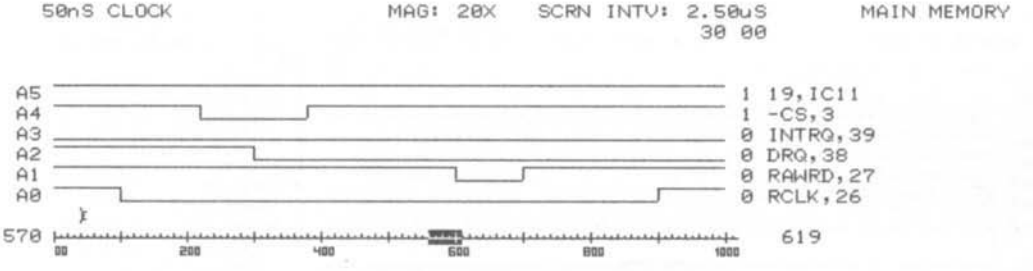


Abb. 7.11.16 Timing beim Lesen von Daten (2,5 µs)

NICOLET PARATRONICS

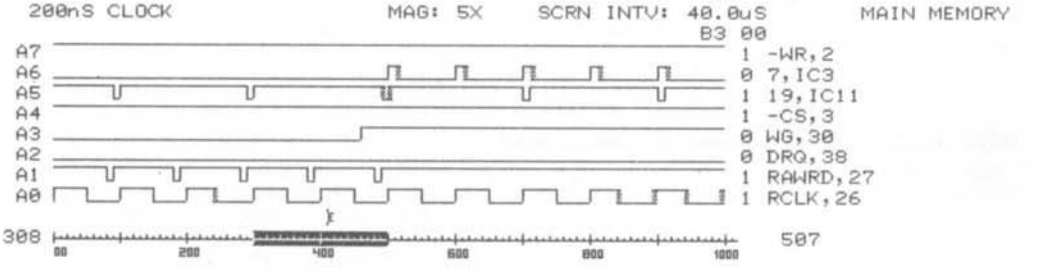


Abb. 7.11.17 Das WG-Signal beim Schreiben

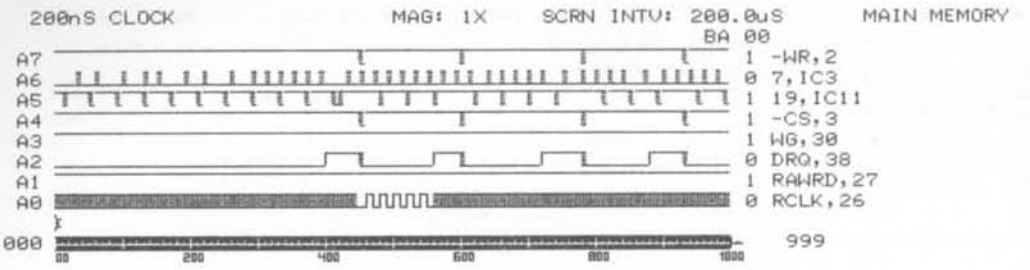


Abb. 7.11.18 Wenn das DRQ-Signal erscheint, werden die Daten übertragen

NICOLET PARATRONICS

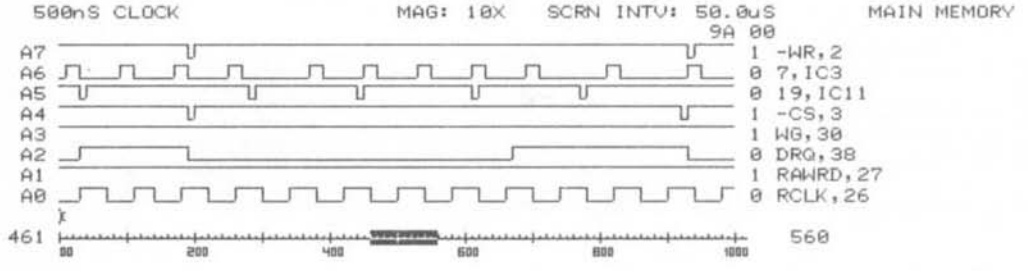


Abb. 7.11.19 Das Schreiben (50 µs)

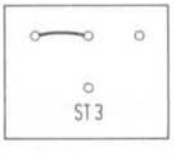


Abb. 7.11.20 Die Brücke ST3, wenn man die automatische Motor-Abschaltung verwendet.
Achtung: Das READY-Signal muß dann vom Laufwerk kommen!

NICOLET PARATRONICS

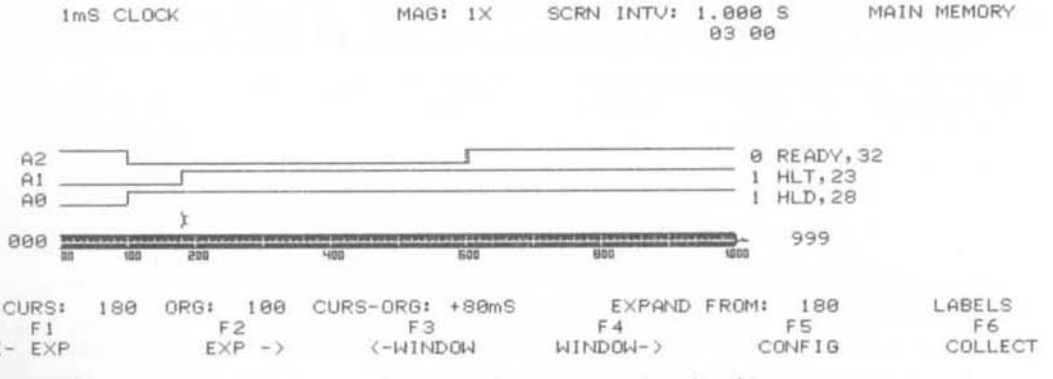


Abb. 7.11.21 Der Start-Vorgang beim Betrieb mit READY-Signal und Motorstart

Will man den Inhalt von EPROMs wieder löschen, benötigt man entweder eine UV-Lampe (es gibt spezielle EPROM-Löschgeräte), oder man begnügt sich am Anfang mit dem Sonnenlicht (das dauert viele Stunden, hoffentlich scheint auch die Sonne, wenn man sie gerade braucht).

Materialliste:

IC1 7406 Inverter mit offenem Kollektor und hoher Kollektor-Spannung
IC2 74121 Monoflop
IC3,IC10 74LS138 1-aus-8 Dekoder
IC4 74LS00 Nand-Glieder
IC5,IC6 74LS245 bidirektionale Bustreiber
IC7 74LS374 Zwischenspeicher mit Tri-State-Ausgang
IC8,IC9 74LS273 Zwischenspeicher mit Löscheingang
IC11 74LS85 Vergleicher
1x 28polige EPROM-Fassung (Zero-Force-Typ)
5x 20polige IC-Fassung
4x 16polige IC-Fassung
3x 14polige IC-Fassung
R1, R2 10 kΩ 1/8 Watt
R3, R6 1 kΩ
R4 100 Ω
R5 10 Ω
R7 330 Ω
Tr1 10 kΩ Helitrimmer
C1,C3 10 µF, Tantal
C2 100 nF
LED1 3mm Leuchtdiode rot
D1 Silizium Diode
T1,T2,T3 BC 107
St1 36polige Stiftleiste, gewinkelt, einreihig.
3x 16poliger DIL-Stecker
1x ges-Leiterplatte PROMMER

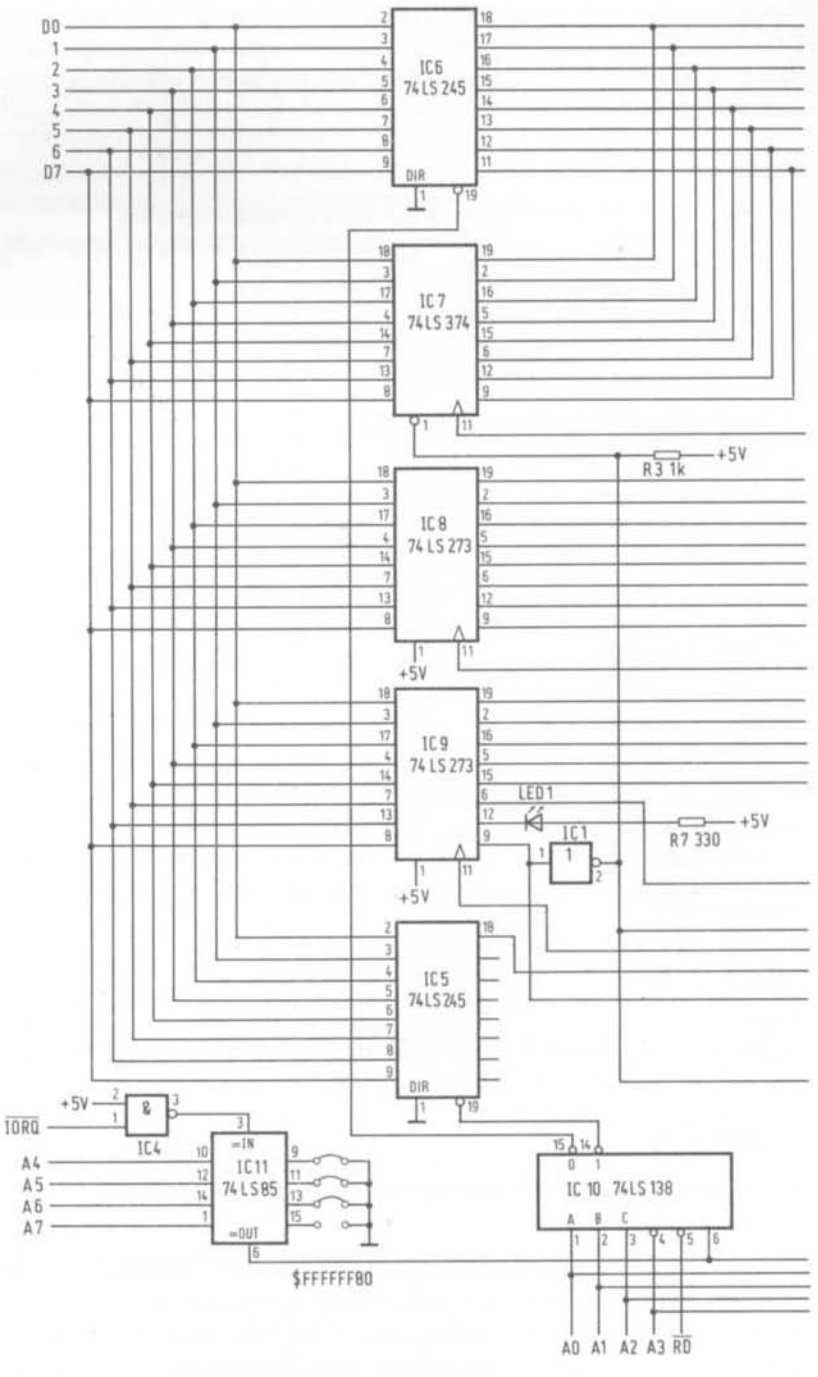
Kenndaten:

Spannungsversorgung: +5 V, 280 mA
+26/22 V, ca. 40 mA
Die +26 V oder +22 V kann man einem gesonderten Netzteil oder einer Wandler-Baugruppe (POW22/26) entnehmen.
Der Prommer ist geeignet für die EPROM-Typen: 2716, 2732, 2732A, 2764, sowie für ähnliche Bauarten, bei geeignetem Anpaß-Stecker.

Zum Löschen der EPROMS:
1x UV-EPROM-Löschgerät.

Diese Geräte gibt es in unterschiedlicher Ausführung im Handel zu kaufen.

Warum braucht man eigentlich ein besonderes Gerät zum Programmieren von EPROMs? EPROMs werden mit Hilfe einer hohen Spannung (21 V oder 25 V) programmiert. Ferner benötigt man besondere Pulse, die dann die Daten in das EPROM übertragen. Die Baugruppe PROMMER beinhaltet alle dazu notwendigen Schaltungsteile. Der Programmiervorgang dauert pro Byte ca. 50 ms.


D0
1
2
3
4
5
6
D7
IC6
74LS245
DIR
IC7
74LS374
IC8
74LS273
IC9
74LS273
R3 1k
+5V
LED1
R7 330
IC1
1
IC5
74LS245
IORQ
IC4
&
A4
A5
A6
A7
IC11
74LS85
=IN
=OUT
$FFFFFF80
IC10 74LS138
A B C
A0 A1 A2 A3 RD

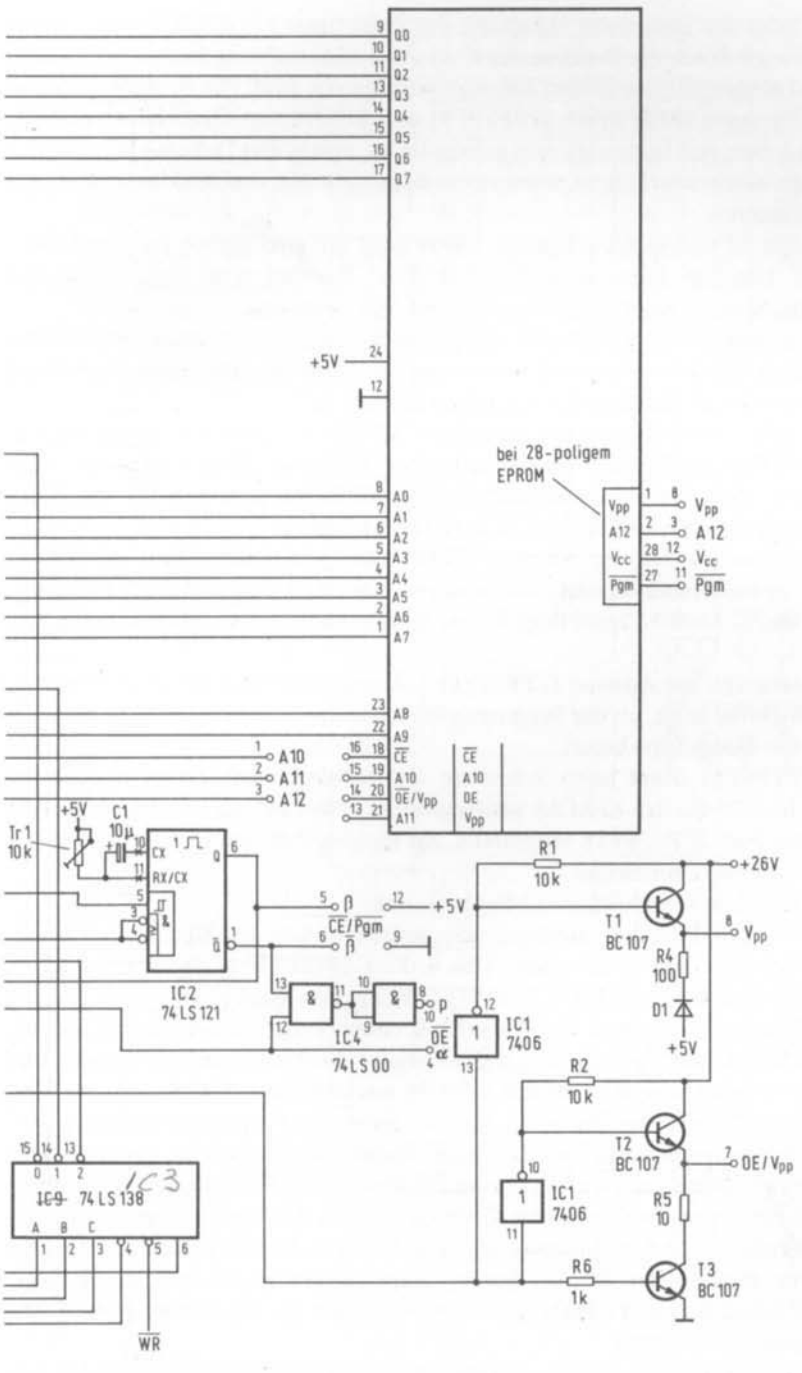


Abb. 7.12.1 Die Schaltung der Baugruppe PROMMER

Abb. 7.12.1 zeigt die komplette Schaltung der Baugruppe PROMMER. Die Adresse der Baugruppe wird durch die Brücken der IC 11-Außenbeschaltung festgelegt. Sie wird auf $FFFFFF80 eingestellt, wozu drei Brücken einzusetzen sind. Die Baugruppe belegt dann die drei Adressen $FFFFFF80, $FFFFFF81 und $FFFFFF82. Über den Dekoder IC 10, der für das Lesen von Informationen zuständig ist, sowie den Dekoder IC 9, der für Schreibvorgänge verantwortlich ist, werden die Bausteine IC 5 und IC 6, bzw. IC 7, IC 8 und IC 9 angesteuert.

Der PROMMER ist universell ausgelegt. Daher sind die wichtigsten Programmiersignale auf eine 16polige Fassung gelegt. Auf diese Fassung wird dann später ein verdrahteter DIL-Stecker gesetzt, der den EPROM-Typ bestimmt.

Auf der Baugruppe befindet sich auch ein Abgleichpunkt, der Trimmer Tr1. Mit ihm muß die Zeitdauer des Monoflops auf 50 ms eingestellt werden. Für diese Einstellung werden wir noch ein Abgleichprogramm kennenlernen.

Abb. 7.12.2 zeigt die Belegung der einzelnen Adressen. Durch das Lesen von der Adresse $FFFFFF80 erhält man den Inhalt eines EPROMs. Dies wird auch dann gebraucht, wenn man die Programmierung eines EPROMs prüfen will. Die Daten gelangen in dieser Betriebsart über das IC 6 an den Datenbus.

Programmierdaten werden zur Adresse $FFFFFF80 geschrieben. Dazu werden die Daten im IC 7 zwischengespeichert. Sie gelangen erst dann an den EPROM-Eingang, wenn an Pin 1 des IC 7 ein 0-Signal liegt. Dieses 0-Signal wird durch einen anderen Port eingestellt.

Durch das Lesen von der Adresse $FFFFFF81 bekommt man eine Status-Information. Bit 0 des Datenwortes zeigt, ob der Programmiervorgang gerade aktiv ist. Nur wenn es auf 0 liegt, ist die Baugruppe bereit.

Der Port $FFFFFF81 dient beim Schreiben als Adreßauswahl. Dabei werden die Leitungen A0 bis A7 an das EPROM weitergereicht. Die Leitungen A8 bis ggf. A12 werden mit dem Port $FFFFFF82 eingestellt. An diesem Port befinden sich auch die Steuersignale zur Programmierung.

Abb. 7.12.3 zeigt die verschiedenen Möglichkeiten.

Wenn man den EPROM-Inhalt auslesen will, so legt man an das Signal „ena" (Bit 7, Port $FFFFFF82) den Wert 0, an „-led" (Bit 6, Port $FFFFFF82) den Wert 1 (dann erlischt die LED) und an „trg" (Bit 5, Port $FFFFFF82) den Wert 0.

In der Abbildung ist darunter der Ablauf bei einem Programmiervorgang gezeigt. Wird das Signal „ena" auf 1 gelegt, erfolgt die Freigabe der Programmierspannung, und die Datenwerte werden vom IC 7 an das EPROM weitergeleitet. Das Signal „trg" löst beim Wechsel von 0 auf 1 das Monoflop aus und leitet den Programmiervorgang ein.

Achtung! Wenn man das Monoflop zu schnell hintereinander auslöst, verändert sich die eingestellte Zeit. Das Monoflop 74121 benötigt eine „Erholzeit". Bei 50 ms Zeitdauer sind das ca. 10 ms. Diese Zeit wird vom Grundprogramm automatisch berücksichtigt.

Wie bereits erwähnt, sind bestimmte Signale auf eine IC-Fassung gelegt, in die kein IC, sondern ein verdrahteter Stecker gesetzt wird. Somit ist es möglich, mit der Baugruppe PROMMER eine Vielzahl von EPROM-Typen zu bearbeiten. *Abb. 7.12.4* zeigt die Belegung der Fassung.

Abhängig vom EPROM-Typ gibt es unterschiedliche Pegel und Polaritäten bei den Programmierpulsen und Spannungen. *Abb. 7.12.5* zeigt drei verschiedene Stecker, mit

	7	6	5	4	3	2	1	Ø	
$FFFFFF8Ø	D7	D6	D5	D4	D3	D2	D1	DØ	Input
$FFFFFF8Ø	D7	D6	D5	D4	D3	D2	D1	DØ	Output
$FFFFFF81	x	x	x	x	x	x	x	busy 1=warten	Input
$FFFFFF81	A7	A6	A5	A4	A3	A2	A1	AØ	Output
$FFFFFF82	ena (OE)	led Ø=an	trg	A12	A11	A10	A9	A8	–

Abb. 7.12.2 Die Belegung der Ports

	ena (α)	led	trg (β)
Lesen	Ø	1	Ø
Program-mieren	1	Ø	Ø
	1	Ø	1
	1	Ø	Ø

Abb. 7.12.3 Die Signale zur Programmierung

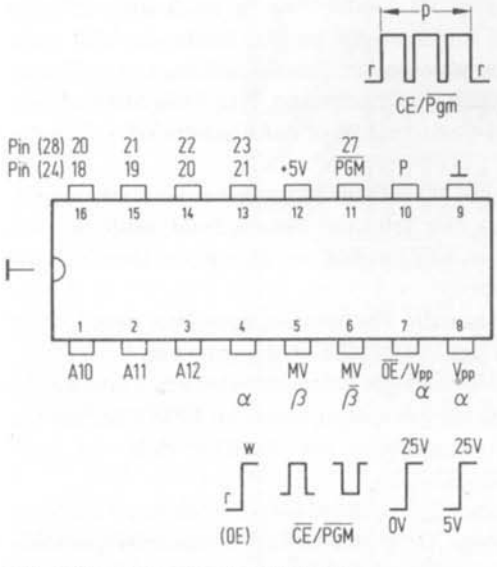


Abb. 7.12.4 Belegung der IC-Fassung

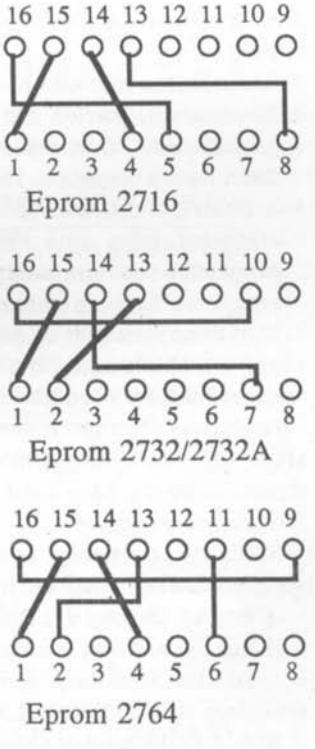


Abb. 7.12.5
Belegung von verschiedenen Steckern

denen sich die gebräuchlichsten EPROMs programmieren lassen. Wer andere EPROMs bearbeiten will, muß sich zunächst ein ausführliches Datenblatt vom Baustein-Hersteller besorgen und dann die passenden Signale am Stecker verdrahten.

Die verschiedenen EPROMs brauchen aber nicht nur eine unterschiedliche Belegung, sondern auch unterschiedliche Spannungen. So benötigen die Bausteine 2716 und 2732 eine Programmierspannung von 25 V. Dazu muß man der Baugruppe 26 V zuführen (1 V fällt in der Schaltung ab). Die EPROMs 2732A und 2764 benötigen eine Spannung von 21 V (also 22 V für die Baugruppe).

Wenn man die Spannung anschließt, sollte man sich vergewissern, daß das EPROM auch mit dieser Spannung programmiert wird. Eine zu hohe Spannung zerstört das EPROM, und bei einer zu niederen wird es nicht programmiert.

Abb. 7.12.6 zeigt die Lötseite, *Abb. 7.12.7* die Bestückungsseite und *Abb. 7.12.8* den Bestückungsplan.

Aufbau und Test:

1. Einlöten aller passiven Bauteile und aller IC-Fassungen. Die drei Brücken sind gemäß dem Bestückungsplan einzusetzen.
2. Einschalten der Versorgungsspannung (+5 V) und prüfen, ob ein Kurzschluß vorliegt.
3. Ausschalten und alle ICs einsetzen.
4. Einschalten, Betrieb mit dem Grundprogramm. Aufruf des Menues „EPROM programmieren". Dort wählt man als Startadresse 0 und als Endadresse $FF, als Ziel kann man 0 eingeben. Die LED muß nun dunkel sein.

   Dann „B" drücken. Die LED leuchtet nun auf. Nach Abschluß der Programmierung erscheint dann eine Fehlermeldung „EPROM Fehler", da ja noch kein EPROM eingesetzt war und auch kein Abgleich durchgeführt wurde. Sollte die LED nicht reagieren, liegt ein Fehler vor. Ggf. sollte man die Brückeneinstellung kontrollieren.
5. Nun kann man sich an den Abgleich machen. Dazu zeigt *Abb. 7.12.9* ein Abgleichprogramm. Nach erfolgter Eingabe sollte es vorsichtshalber auf Kassette oder Diskette abgespeichert werden.

Nach dem Start erscheint die Pulsbreite des Monoflops in ms mit einer Nachkommastelle auf dem Bildschirm. Wenn man nun den Trimmer Tr1 verdreht, muß sich die Anzeige ändern. Man kann dann einen Wert von ca. 50,0 ms einstellen. Damit ist die Baugruppe abgeglichen.

Sollte keine Anzeige erscheinen, muß man die Programmeingabe mit dem Listing vergleichen. Ggf. liegt auch ein Fehler auf der PROMMER-Baugruppe vor.

Abb. 7.12.10 zeigt das Pulsdiagramm des Monoflops. Zunächst wird Pin 3 und 4 auf 0 V gelegt, gleichzeitig gelangt damit die Programmierspannung an das EPROM. Über Pin 5 wird das Monoflop getriggert. An Pin 6 erscheint ein positiver Puls, der nach erfolgtem Abgleich 50 ms lang sein muß.

Abb. 7.12.11 zeigt die Programmierung eines Bytes. Dabei war die Adresse 0 als Start, sowie 0 als Ende und 0 als Ziel angegeben. Dann wird die Programmierspannung nämlich gleich nach 50 ms wieder zurückgenommen.

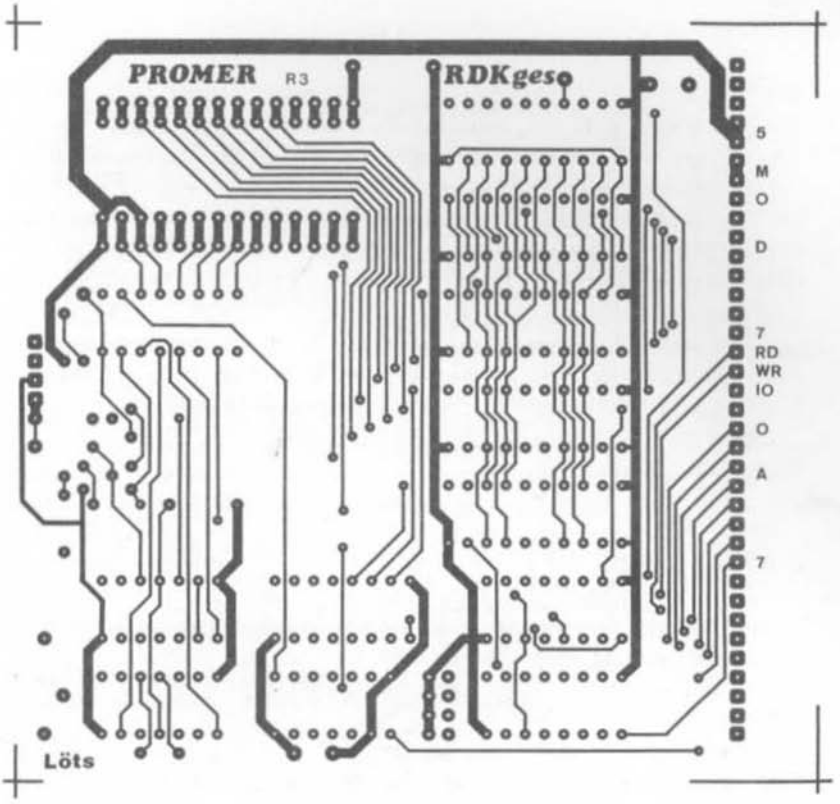


Abb. 7.12.6
Lötseite
der Baugruppe
PROMMER

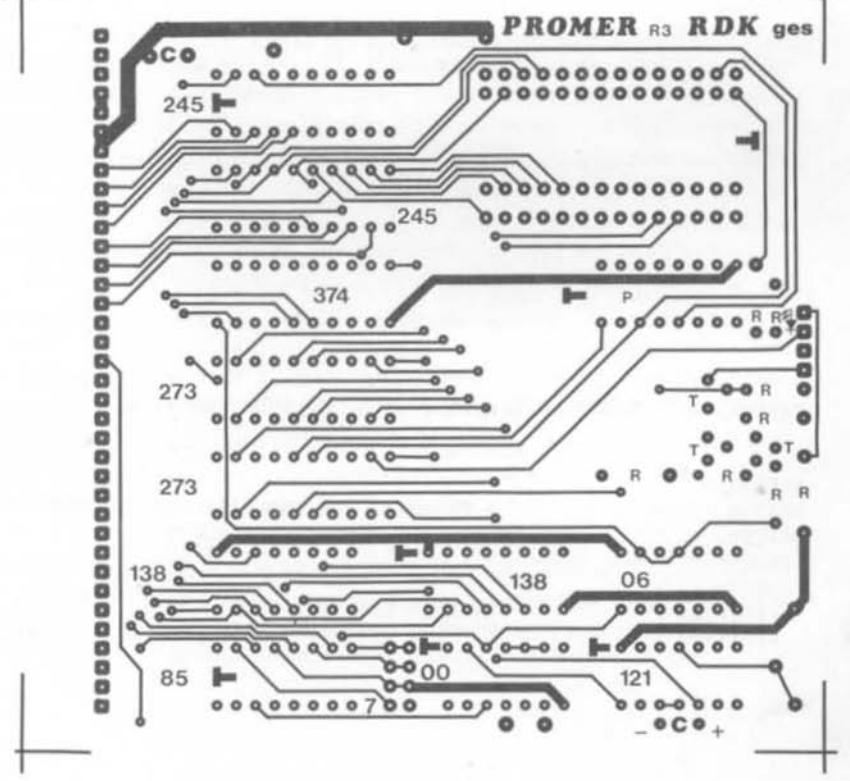


Abb. 7.12.7
Bestückungs-
seite der
Baugruppe
PROMMER

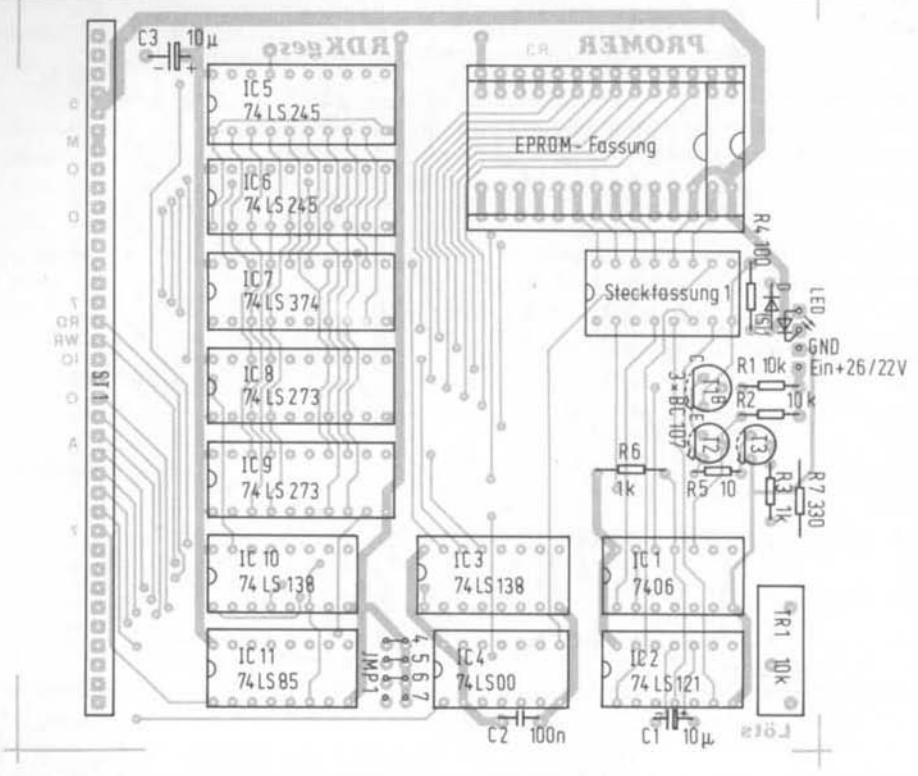


Abb. 7.12.8 Bestückungsplan der Baugruppe PROMMER

```
Rolf-D.Klein 68000/08 Assembler 4.3 (C) 1984, Seite 1

0E9C00
0E9C00                  * PROGRAMM ZUM ABGLEICH DER PROMMER-BAUGRUPPE
0E9C00                  * 850604 ROLF-DIETER KLEIN
0E9C00                  *
0E9C00
= FFFFFF80              PROMMER EQU $FFFFFF80
= FFFFFF70              GDP EQU $FFFFFF70                    zu Abb. 7.12.9
```

```
0E9C00
0E9C00                      START:
0E9C00  4EB9 000E0DA0        JSR @CLR        * BILDSCHIRM LOESCHEN
0E9C06  4240                 CLR D0
0E9C08  4241                 CLR D1
0E9C0A  4EB9 000E0E0C        JSR @NEWPAGE    * BILDSEITE 0 NEHMEN
0E9C10                      SCHLEIFE:
0E9C10  4EB9 000E9C6C        JSR MESSE20     * 20MS REFERENZ NACH D4.L
0E9C16  4EB9 000E9C9A        JSR MESSEPROM   * MESSEWERT NACH D5.L
0E9C1C  CAFC 00C8            MULU #200,D5    * BERECHNUNG MIT 16 BIT AUSREICHEND.
0E9C20  8AC4                 DIVU D4,D5      * D5.W IST ERGEBNIS, IN MILLISEKUNDEN*10
0E9C22  41F9 000E9CC9        LEA BUFFER,A0
0E9C28  3005                 MOVE D5,D0      * DANN DEZIMAL AUSGEBEN
0E9C2A  4EB9 000E15FC        JSR @PRINT4D    * ZUNAECHST IN DEN BUFFER
0E9C30  10E8 FFFF            MOVE.B -1(A0),(A0)+ * KOMMASTELLE
0E9C34  117C 002E FFFE       MOVE.B #'.',-2(A0)  * DORT KOMMA ABLEGEN
0E9C3A  43F9 000E9CC0        LEA TEXT,A1     * DANN TEXT DAHINTER
0E9C40                      SCHL1:
0E9C40  10D9                 MOVE.B (A1)+,(A0)+
0E9C42  6600 FFFC            BNE SCHL1
0E9C46  4210                 CLR.B (A0)      * DANN AUSGEBEN
0E9C48  41F9 000E9CC9        LEA BUFFER,A0
0E9C4E  323C 000A            MOVE #10,D1     * X
0E9C52  343C 0064            MOVE #100,D2    * Y
0E9C56  103C 0033            MOVE.B #$33,D0 * SCHRIFTGROESSE
0E9C5A  4EB9 000E1386        JSR @WRITE
0E9C60  4EB9 000E09D8        JSR @CSTS
0E9C66  6700 FFA8            BEQ SCHLEIFE  * IMMER NEU MESSE20 AUFRUFEN,
0E9C6A                                     * DA DAS MONOFLOP ERHOLZEIT BENOETIGT.
0E9C6A  4E75                 RTS
0E9C6C
0E9C6C                      MESSE20:        * REFERENZ MESSEN
0E9C6C  4284                 CLR.L D4        * DORTHIN ERGEBNIS
0E9C6E  41F9 FFFFFF70        LEA GDP,A0
0E9C74                      MES0:
0E9C74  0810 0001            BTST #1,(A0)    * STARTFLANKE
0E9C78  6700 FFFA            BEQ MES0        * ZUM SYNCHRONISIEREN
0E9C7C                      MES1:
0E9C7C  0810 0001            BTST #1,(A0)
0E9C80  6600 FFFA            BNE MES1
0E9C84                      MES2:           * DANN GANZE PERIODE MESSEN
0E9C84  5284                 ADDQ.L #1,D4    * LOW UND HIGH-TEIL ZUSAMMEN
0E9C86  0810 0001            BTST #1,(A0)    * DAUERN 20MS
0E9C8A  6700 FFF8            BEQ MES2        * TEIL1
0E9C8E                      MES3:
0E9C8E  5284                 ADDQ.L #1,D4    * ACHTUNG, QUICK-BEFEHL
0E9C90  0810 0001            BTST #1,(A0)
0E9C94  6600 FFF8            BNE MES3        * 1-TEIL AUCH MESSEN
0E9C98  4E75                 RTS
0E9C9A
0E9C9A
0E9C9A                      MESSEPROM:      * PROMMER TRIGGERN UND MESSEN
0E9C9A  4285                 CLR.L D5        * DORT ERGEBNIS
0E9C9C  41F9 FFFFFF82        LEA PROMMER+2,A0 * IST TRIGGERPORT
0E9CA2  43F9 FFFFFF81        LEA PROMMER+1,A1 * IST ABFRAGEPORT
0E9CA8  10BC 0080            MOVE.B #$80,(A0) * LED AN ALS WARNUNG
```

zu Abb. 7.12.9

```
Rolf-D.Klein 68000/08 Assembler 4.3 (C) 1984, Seite 2

0E9CAC  10BC 00A0        MOVE.B #$A0,(A0) * UND START
0E9CB0                  MESP1:
0E9CB0  5285             ADDQ.L #1,D5    * ACHTUNG QUICK-BEFEHL
0E9CB2  0811 0000        BTST #0,(A1)
0E9CB6  6600 FFF8        BNE MESP1       * NUR HIGH-TEIL MESSEN
0E9CBA  10BC 0040        MOVE.B #$40,(A0) * UND STOP.
0E9CBE  4E75             RTS
0E9CC0
0E9CC0  206D7320202020  TEXT: DC.B ' ms      ',0 * LEERZEICHEN WICHTIG.
0E9CC7  2000
0E9CC9                  BUFFER: DS.B 80
0E9D19
0E9D19
0E9D19                   END

0E8B5C  Ende-Symboltabelle
00993C  Ende-Debug-Tabelle
```

Abb. 7.12.9 Das Abgleichprogramm

E9573

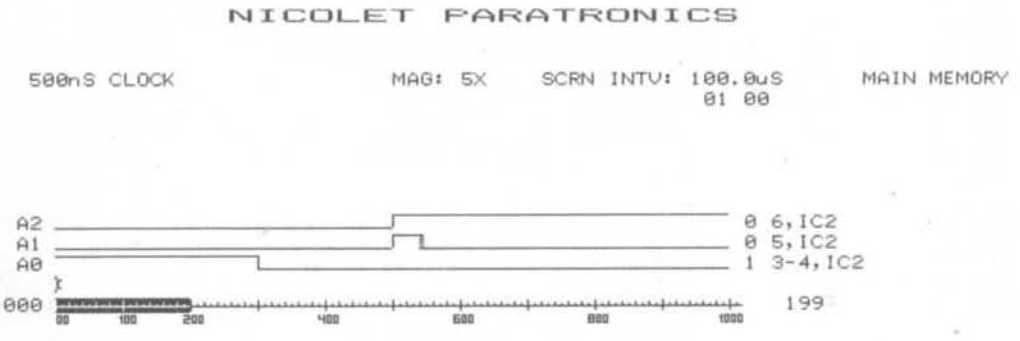


Abb. 7.12.10 Das Monoflop wird ausgelöst

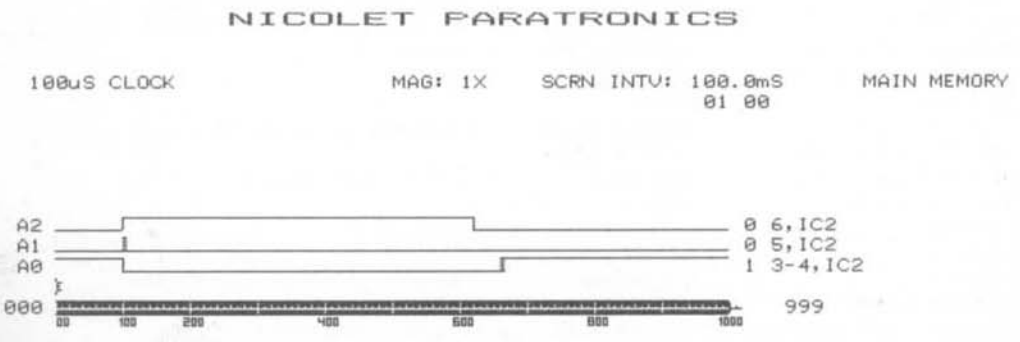


Abb. 7.12.11 Die Programmierung eines Bytes

Hinweis zum Grundprogramm: Bei der Programmierung werden drei Adressen abgefragt. Die „von"-Adresse ist die Adresse im Hauptspeicher, von wo an die Übertragung ins EPROM beginnen soll. Die „bis"-Adresse ist die letzte Adresse, die noch programmiert werden soll. Die „nach"-Adresse ist die erste Adresse im EPROM, bei der die Programmierung begonnen werden soll. Normalerweise wird sie mit 0 angegeben. Nur dann, wenn mehrere Programme in einem EPROM unterzubringen sind, wird man bei einer höheren Adresse anfangen, die Daten unterzubringen.

Beim Lesen von EPROMs ist es umgekehrt. Die „von"-Adresse ist die erste Adresse im EPROM, bei der begonnen wird, das EPROM auszulesen. Die „bis"-Adresse ist die letzte Adresse im EPROM, bis zu der ausgelesen wird, und die „nach"-Adresse ist die erste Adresse im Hauptspeicher.

7.13 Die IOE-Baugruppe

Eine der wichtigsten Baugruppen im ganzen System ist die IOE-Baugruppe. Damit ist es dem Computer möglich, mit der Außenwelt in Verbindung zu treten. So kann man die IOE-Baugruppe zum Anschluß von Tastern, Lampen, Schaltern oder Treibern für Motoren etc. verwenden.

Materialliste:

IC 1 74LS04 Inverter
IC 2, IC 3 74LS245 bidirektionaler Bustreiber
IC 4, IC 5 74LS374 Zwischenspeicher mit TRI-State-Ausgang
IC 6 74LS139 zwei 1-aus-4 Dekoder
IC 7 74LS85 Vergleicher
IC 8 74LS32 Oder-Glied
4x 20polige IC-Fassung
2x 16polige IC-Fassung
2x 14polige IC-Fassung
C1 10 µF
1x 36polige Stiftleiste, gewinkelt, einreihig.
1x ges-IOE-Leiterplatte.
ggf.:
1x 50polige doppelreihige Stiftleiste, gerade.
Kenndaten:
Spannungsversorgung: +5 V, 190 mA

Abb. 7.13.1 zeigt den Schaltplan der Baugruppe.

Die IOE-Baugruppe besitzt zwei 8-Bit-Eingabe-, sowie zwei 8-Bit-Ausgabeeinheiten. Als Eingabeeinheit dient der Baustein 74LS245 (IC 2 und IC 3), und für die Ausgabe tut je ein 74LS374 (IC 4 und IC 5) seinen Dienst.

Die Adresse der IO-Ports (Ein-/Ausgabeeinheiten) wird mit Hilfe des IC 7 (74LS85) bestimmt. In seiner Beschaltung befinden sich vier Brücken, mit denen man die Adresse einstellen kann. Der Vergleicher IC 7 liefert an seinem Ausgang Pin 6 nur dann ein 1-Signal, wenn das Datenmuster an den Pins 9, 11, 14 und 1 mit dem an den Pins 10, 12, 13 und 15 übereinstimmt.

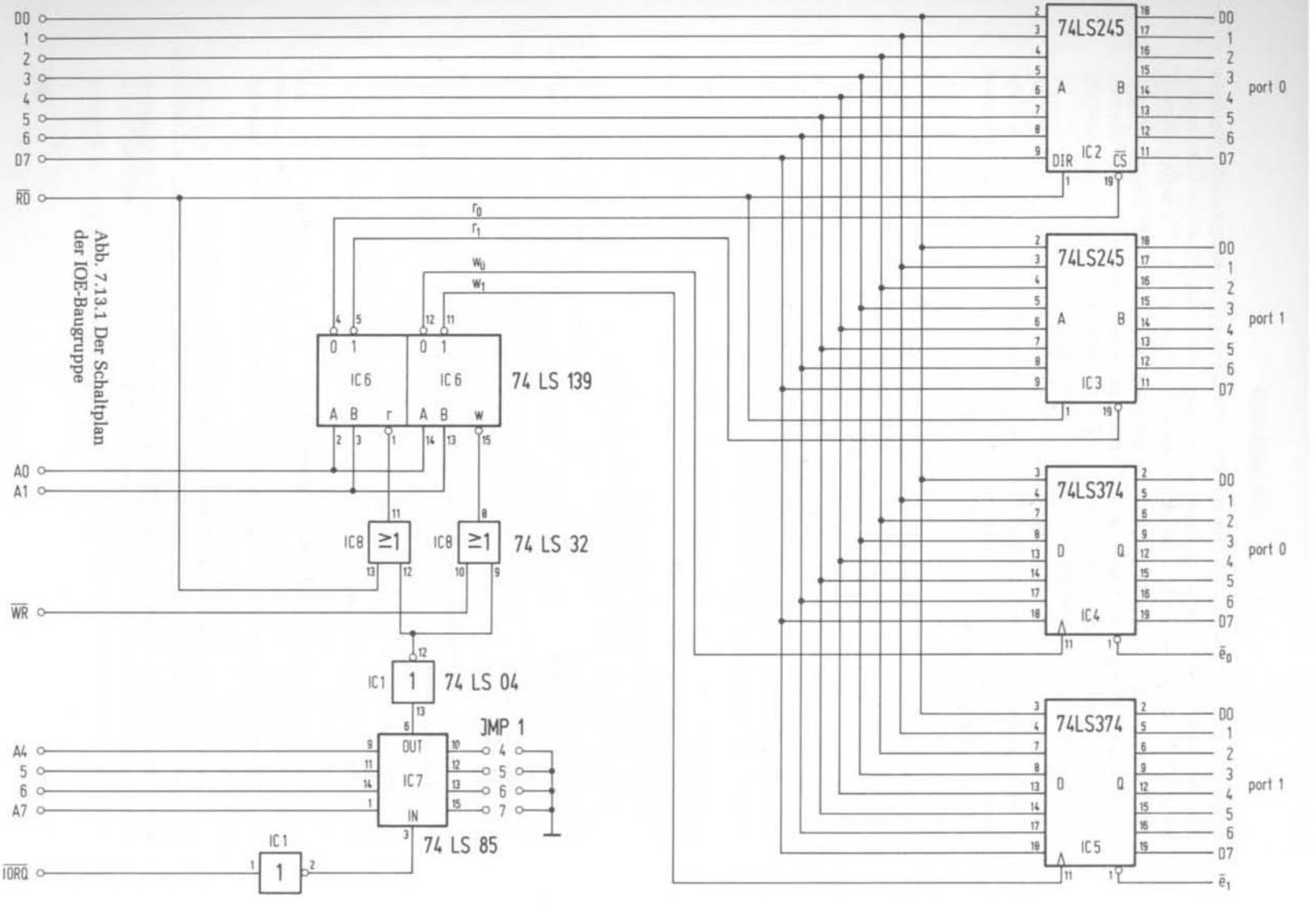


Abb. 7.13.1 Der Schaltplan der IOE-Baugruppe

Ferner muß an Pin 3, dem „="-Eingang, ein 1-Signal anliegen. Dieser Eingang ist mit dem Ausgang eines Inverters verbunden. Der Eingang des Inverters wird vom IORQ-Quer-Signal des Busses bedient. Also reagiert der Ausgang Pin 6 des Vergleichers nur dann, wenn die Adresse an A4 bis A7 mit den Signalpegeln der Brücken 4 bis 7 (JMP1) übereinstimmt und ein IO-Zugriff vorliegt (IORQ-Quer ist dann auf 0).

Diese Information gelangt an die beiden Oder-Glieder (IC 8). Je nachdem, ob ein Lese- oder ein Schreibzugriff stattfindet, wird einer der beiden Dekoder (IC 6) angesteuert. Bei einem Lesezugriff wird der linke (IC 6, Pin 1), bei einem Schreibzugriff dagegen der rechte Dekoder (IC 6, Pin 15) freigegeben. Der Dekoder kann nun nochmals von vier Adressen eine auswählen; er ist dazu mit den Adreßleitungen A0 und A1 verbunden. Da die Adreßleitungen A2 und A3 nicht verbunden sind, ist die Adressierung nicht eindeutig. Diese Leitungen kann man aber z. B. auf dem Lochrasterfeld für eigene Schaltungen verwerten. Die Ausgänge 2 und 3 (Pin 6, 7 und 10, 9 des IC 6) sind ebenfalls nicht verbunden. Sie sind jedoch im Layout an je zwei Löcher geführt. Mit diesen Signalen kann man die Anzahl der IO-Einheiten bei Bedarf erweitern.

Die Bausteine IC 2 und IC 3 sind bidirektionale Bustreiber. Das heißt, die Daten können in beide Richtungen übertragen werden. Diese Bausteine wurden gewählt, um die Baugruppe möglichst universell zu gestalten. Wird das Signal an Pin 19 nicht nur bei einem Lese-Zugriff, sondern auch beim Schreiben auf 0 gelegt, kann man Daten in beiden Richtungen übertragen.

Als Ausgangsbausteine wurden 74LS374 verwendet. Dies sind Zwischenspeicher mit TRI-State-Ausgängen. Dazu besitzen die Bausteine einen Freigabeeingang (Pin 1, IC 4, IC 5, bzw. e0-Quer, e1-Quer).

Wenn an diesem Eingang ein 0-Signal liegt, arbeiten die Ausgänge der Zwischenspeicher (Q-Seite) wie normale TTL-Bausteine. Wenn aber der Eingang Pin 1 ein 1-Signal führt, sind die Ausgänge offen. Diese Eigenschaft kennen wir bereits von anderen Bausteinen am Datenbus. Der Ausgang ist hochohmig und beeinflußt somit den Bus nicht. Durch diese Eigenschaft ist es möglich, andere Ausgänge parallel zu schalten, die ebenfalls TRI-State-Zustände haben können. Es darf dann immer nur einer dieser Ausgänge einen definierten 1- oder 0-Pegel annehmen.

Abb. 7.13.2 zeigt die Belegung der Anschlüsse auf dem Lochrasterfeld. Das Lochrasterfeld ist von der Bestückungsseite her betrachtet. Oben liegt ein Masseanschluß (M). Anschließend liegen die Anschlüsse von Port 0 (IC 2), danach folgt Port 1 (IC 3), dann der Ausgang Port 1 (IC 5) und zum Schluß der Ausgang Port 0 (IC 4). Ganz unten sind die beiden Freigabeeingänge e0-Quer und e1-Quer angebracht. Darunter befinden sich nochmals Masseanschlüsse.

Für den normalen Betrieb werden Brücken von e0-Quer nach Masse und von e1-Quer nach Masse gelötet.

Die IOE-Baugruppe wird im Verlauf des Buches mit unterschiedlichen Adressen verwendet. *Abb. 7.13.3* zeigt alle möglichen Brückeneinstellungen mit der jeweiligen IO-Adresse. Dabei wird die Adresse 40 z. B. mit $FFFFFF40 angesprochen. Auch ist nur die erste Adresse angegeben. Es werden jeweils 16 aufeinanderfolgende Adressen belegt, also z. B. bei A0 die Adresse $FFFFFFA0 bis $FFFFFFAF. Port 0 liegt dann auf Adresse $FFFFFFA0 und Port 1 auf Adresse $FFFFFFA1. Da nicht alle IO-Adressen in die Dekodierung mit einbezogen sind, tauchen die Ports aber auch auf den weiteren

Adressen auf, also Port 0 bei $FFFFFFA4, $FFFFFFA8, $FFFFFFAC und Port 1 bei $FFFFFFA5, $FFFFFFA9 und $FFFFFFAD.

Abb. 7.13.4 zeigt die Lötseite und *Abb. 7.13.5* die Bestückungsseite der Baugruppe. In *Abb. 7.13.6* ist der Bestückungsplan abgedruckt.

Aufbau und Test:

1. Einlöten aller passiven Bauteile und der IC-Fassungen.
2. Prüfen, ob ein Kurzschluß vorhanden ist, indem die Baugruppe in der jetzigen Version in den Bus gesteckt wird. Das Grundprogramm muß sich noch melden.
3. Ausschalten und Einsetzen der IC-Bausteine.
4. Einlöten aller Brücken 4, 5, 6, 7. Damit erhält die Baugruppe die Adresse $FFFFFF00.
5. Einlöten der beiden Brücken e0-Quer und e1-Quer nach Masse (0V).
6. Einschalten. Zum Test wählt man das Menue IO-lesen. Als Adresse gibt man zuerst $FFFFFF00 ein.
7. Nun muß auf der Anzeige 11111111 und FF erscheinen. Man drückt die Taste „D" für Dauerfunktion. Nun kann man mit einem Stück Draht an die Eingänge von Port 0 gehen und diese mit 0 V verbinden (Vorsicht, nichts Anderes berühren).

   Auf der Anzeige muß die Position des Bits jeweils mit 0 gekennzeichnet werden. Verbindet man z. B. Bit 4 von Port 0 mit 0 V, muß die Bitfolge 11101111 erscheinen. So kann man alle Eingänge prüfen. Das gleiche gilt für den Port 1 mit der Adresse $FFFFFF01.
8. Test der Ausgänge. Dazu verwendet man einen Prüfstift.

   Man wählt das Menue IO-setzen. Dort wählt man zunächst die Adresse $FFFFFF00 aus. Als Datenwert versuchen wir zunächst mal den Wert 0. Mit dem Prüfstift kann man nun die Ausgänge kontrollieren. Dann kann man mit dem Menue den Wert $FF an den Port 0 ($FFFFFF00) ausgeben und wieder messen. Das gleiche wiederholt man mit dem Port 1 auf Adresse $FFFFFF01.

Dieser Test prüft noch keine Kurzschlüsse an den Ausgängen. Man könnte das tun, indem man einfach alle Werte von 0 bis $FF an die Ports ausgibt (bzw. 0 bis 255 dezimal); schneller geht es mit einem kleinen Programm:

```
TEST: MOVE.B D0,$FFFFFF00
      ADD.B´,D0
      BRA TEST
```

Mit einem Skop kann man dann die Ausgänge des Ports beobachten. Wenn alles gut geht, muß an jedem Ausgang ein symmetrischer Takt liegen. Dieser Takt hat an Bit 0 die höchste und an Bit 7 die niedrigste Frequenz.

Wenn man $FFFFFF01 in der ersten Anweisung einsetzt, kann man auch den zweiten Port testen.

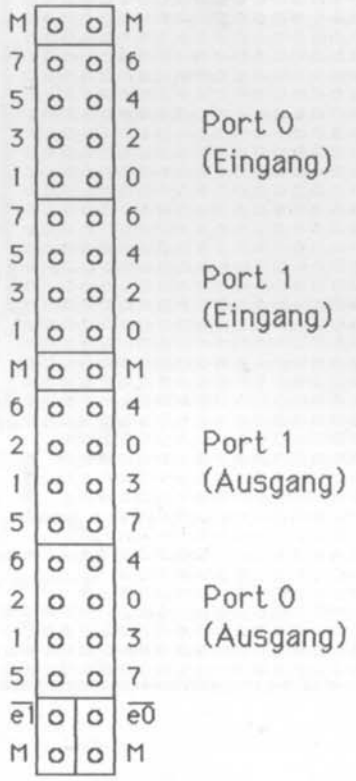


Abb. 7.13.2
Die Anschluß-Belegung
am Lochrasterfeld

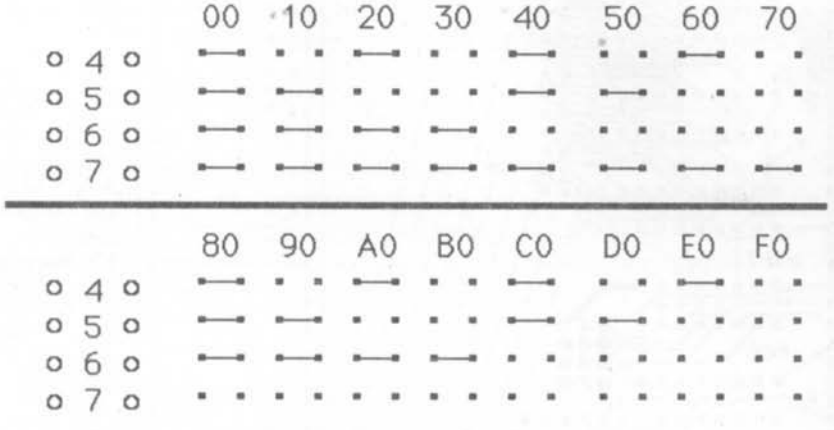


Abb. 7.13.3 Die Einstellung von IO-Adressen

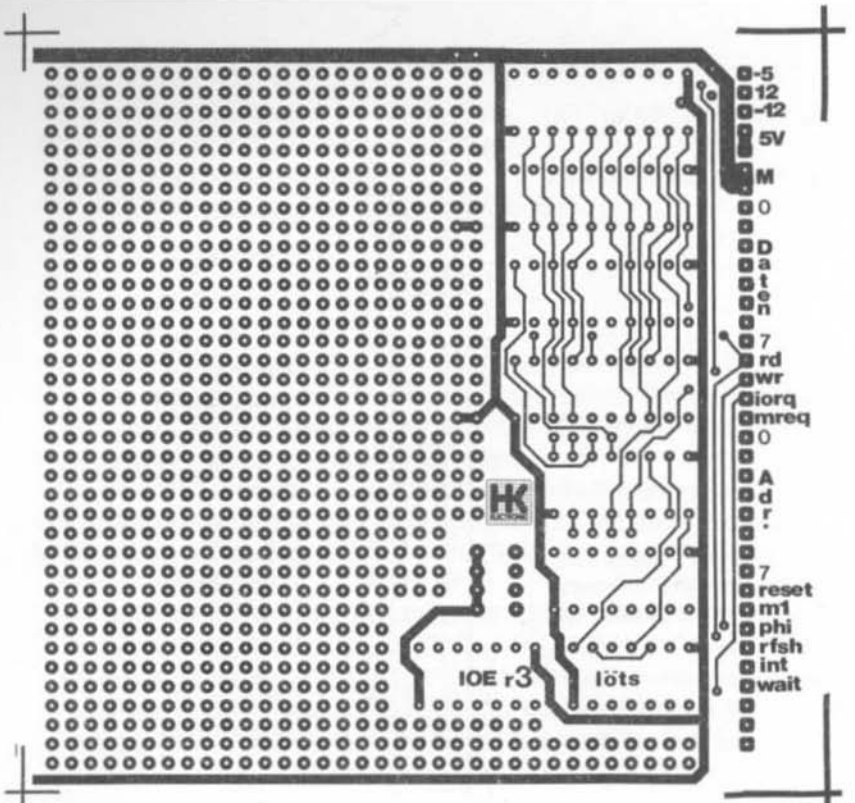


Abb. 7.13.4
Die Lötseite
der IOE-
Baugruppe

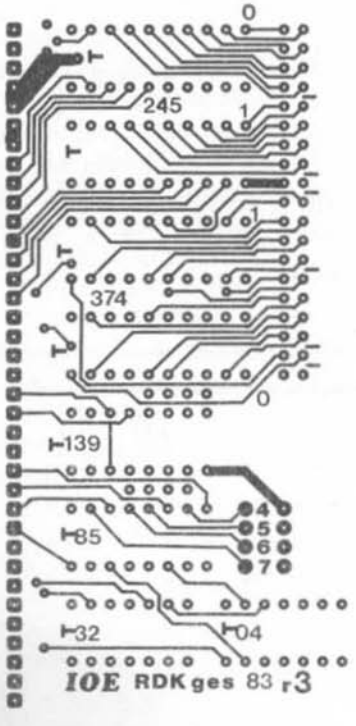


Abb. 7.13.5
Die Bestückungs-
seite der
IOE-Baugruppe

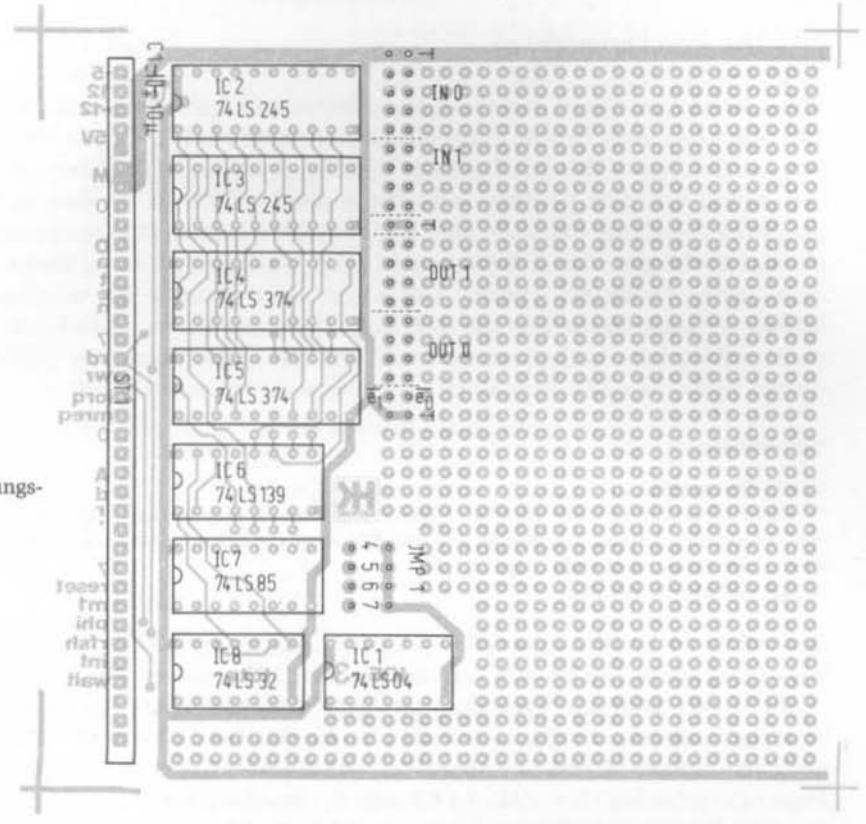


Abb. 7.13.6
Der Bestückungsplan der IOE-Baugruppe

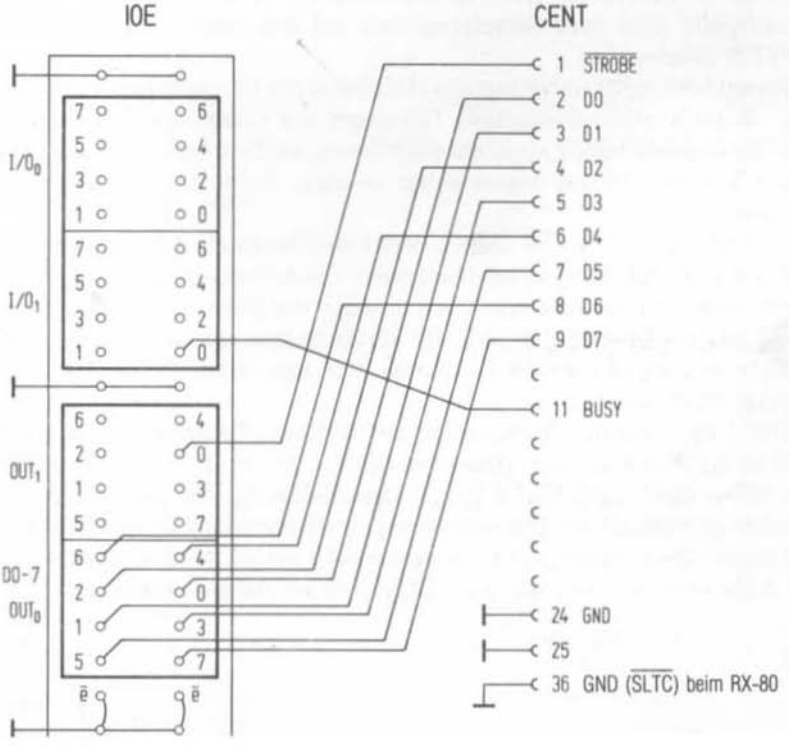


Abb. 7.14.1
Verbindungsschema

7.14 Der Druckeranschluß

Wenn man so richtig mit dem Computer arbeiten will, kommt man um die Anschaffung eines Druckers nicht herum. Dabei gibt es im Prinzip zwei Arten von Druckeranschlüssen, den Centronics-Anschluß und den seriellen Anschluß. Wir wollen uns der Einfachheit halber mit dem Centronics-Anschluß beschäftigen. Es handelt sich um einen parallelen Anschluß, das heißt, alle Daten werden zu einem Byte zusammengefaßt und parallel übertragen. Die IOE-Baugruppe kann zur Verwirklichung dieses Centronics-Anschlusses verwendet werden. Wer schon einen Drucker mit seriellem Anschluß besitzt, der kann diesen über die SER-Baugruppe anschließen (siehe Buch „Mikrocomputer selbstgebaut und programmiert", und die Unterprogramme „SIINIT, SO" in diesem Buch). Die SER-Baugruppe wird hier nicht weiter behandelt.

Materialliste:

1x IOE-Baugruppe
1x Centronics-Buchse oder Stecker 36polig (Fachhandel fragen). Eine Buchse, wenn man ein konfektioniertes Verlängerungskabel mit zwei Steckern zusätzlich verwendet, einen Stecker, wenn man diesen direkt verdrahten will.
1x ggf. Verlängerungskabel.

Der Aufbau der Schaltung gestaltet sich ganz einfach, wenn man schon eine fertige IOE-Baugruppe besitzt. Im Prinzip sind nur ein paar Leitungen zu legen.

Abb. 7.14.1 zeigt das Verdrahtungsschema. Die Anschlüsse der 36poligen Verbindung des Centronics-Steckers sind von 1 bis 36 numeriert. Man verbindet die Leitungen nach dem Schema. Dabei ist zu beachten, daß die Anschlüsse e0-Quer und e1-Quer mit Masse zu verbinden sind. *Abb. 7.14.2* zeigt die Brückeneinstellung der IOE-Baugruppe. Sie sorgt für eine Dekodierung im Bereich $FFFFFF40 bis $FFFFFF4F. Die Centronics-Schnittstelle wird vom Grundprogramm auf den beiden Adressen $FFFFFF48 und $FFFFFF49 adressiert.
Achtung! Man sollte unbedingt das Handbuch des Druckers genau lesen, um festzustellen, ob auch alle notwendigen Leitungen der Centronics-Verbindung belegt sind. Manche Drucker haben noch ein paar besondere Leitungen. So muß beim RX-80 von Epson z.B. Pin 36 auf Masse gelegt werden, damit der Drucker überhaupt etwas annimmt.

Die Leitungen D0 bis D7 dienen dem Datentransport. Dabei werden die Zeichen im ASCII (siehe KEY-Baugruppe) übertragen. Die Leitung Strobe-Quer sagt dem Drucker durch einen kurzen Puls von 1 auf 0, wann die Daten gültig sind. Über die Leitung BUSY teilt der Drucker mit, ob er für die Aufnahme von weiteren Zeichen bereit ist; er benötigt ja zum Druck eine bestimmte Zeit. Das Signal BUSY liegt auf 1, wenn der Drucker nicht bereit ist.

Abb. 7.14.3 zeigt die Übertragung eines Zeichens. Zunächst werden die Daten von der CPU an das Port 0 gesandt. Daher erscheint an Pin 11, IC 5 ein kurzer Puls. Dann wird das Strobe-Quer-Signal auf 0 gelegt. Unmittelbar danach geht BUSY auf 1, denn der Drucker ist nun mit der Übernahme und Weiterverarbeitung des Zeichens beschäftigt. Das Strobe-Quer-Signal geht dann wieder auf 1 zurück. Nach einer Weile (nicht mehr in der Aufnahme sichtbar) fällt auch BUSY wieder auf 0 zurück.

Abb. 7.14.2
Die Brückeneinstellung

NICOLET PARATRONICS

200nS CLOCK MAG: 20X SCRN INTV: 10.0uS MAIN MEMORY
06 00

A2 1 11, IC5
A1 1 BUSY
A0 0 -STROBE
076 125
00 100 200 400 600 800 1000

CURS: 103 ORG: 100 CURS-ORG: +600nS EXPAND FROM: 077 LABELS
F1 F2 F3 F4 F5 F6
<- EXP EXP -> <-WINDOW WINDOW-> CONFIG COLLECT

Abb. 7.14.3 Die Übertragung eines Bytes

10uS CLOCK MAG: 5X SCRN INTV: 2.00mS MAIN MEMORY
05 00

A2 1 11, IC5
A1 0 BUSY
A0 1 -STROBE

Abb. 7.14.4 Die Übertragung von Daten

Abb.7.14.4 zeigt eine verkleinerte Darstellung bei der aufeinanderfolgenden Übertragung von Zeichen. Man sieht, daß nach der Freigabe des BUSY-Signals durch den Drucker nur noch kurze Zeit bis zur Übertragung des nächsten Zeichens vergeht. Gleich darauf meldet dann der Drucker erneut BUSY.

Kommt keine Übertragung bzw. kein Druck zustande, sollte man diese Signale überprüfen. Wichtig ist das Signal BUSY. Es muß auf 0 V liegen, wenn man keine Zeichen überträgt. Ist das nicht der Fall, sollte man am Drucker prüfen, ob er selektiert ist (meist eine Kontrollampe und eine Taste zum Selektieren) und ob Papier eingelegt ist. Außerdem tut man gut daran, die Kabel zu überprüfen.

Wenn das Handshake, also das Hin-und-her-Spiel der Signale funktioniert und trotzdem keine vernünftigen Zeichen ankommen, sollte man kontrollieren, ob nicht vielleicht Kabel vertauscht sind.

Weitere Baugruppen finden Sie im Buch „Mikrocomputer selbstgebaut und programmiert", sowie bei den Herstellern der Leiterplatten und Bausätze des NDR-KLEIN-Computers.

Anhang

Im Anhang finden Sie das Listing des Formatierers.

Nicht abgedruckt sind folgende Listings:
- Pascal/S-Compiler,
- Gosi-Compiler,
- Grundprogramm,
- Hexdump des Grundprogramms,
- Hexdump des Formatierers.

Ein Abdruck dieser Listings hätte den Rahmen des Buches gesprengt. Wenn Sie daran interessiert sind, können Sie das entsprechende Listing beim

Franzis-Verlag, Abteilung: Software-Service bekommen. Ebenso sind dort alle EPROMs mit den fertigen Programmen und Disketten mit dem CP/M-68k erhältlich.

Außerdem sind die Beschreibungen für
- die Befehle des Grundprogramms,
- Pascal/S-Befehle,
- Gosi-Befehle ebenfalls beim Franzis-Software-Service sowie bei den Lieferanten der Bausätze erhältlich.

A Listing des Universal-Formatierers

Die folgende Abbildung zeigt das Listing des Universal-Formatierers. Hier wurde nur ein Modul verwendet, daher stimmen die Adressen bereits mit der EPROM-Version überein.

```
...A68K 4.0a 06/13/83   ...Run on 04/29/85
                         1.
                         2. ********************************************
                         3. * Universal Formatierer fuer den NDR-68K-Computer *
                         4. * (C) 1985 Rolf-Dieter Klein   V1.0  850108      *
                         5. ********************************************
                         6.
                         7. * Verschiebbare Form, mit Bibliotheksfunktion
                         8. * ca. 8K RAM wird verwendet.
                         9.
     00000001           10. cpu equ 1              * 1=68008/2=68000/4=68020
                        11.
```

```
      FFFFFFC0          12. flo0 equ cpu%$ffffffc0
      FFFFFFC1          13. flo1 equ cpu%$ffffffc1
      FFFFFFC2          14. flo2 equ cpu%$ffffffc2
      FFFFFFC3          15. flo3 equ cpu%$ffffffc3
      FFFFFFC4          16. flo4 equ cpu%$ffffffc4
                        17. *
                        18.
000000                  19.  offset 0               * Variablenspeicher Global,
                        20.                         * verschiebbar bezogen auf 0
                        21.
                        22. * Wort Variable
      00000000          23. oldstack:       ds.l 1  * Stackpointer Merker
      00000004          24. anzbyte:        ds.w 1  * Anzahl Bytes pro Sektor
                        25. * Byte Variable
      00000006          26. chars:          ds.b 1  * Zeichenzaehler fuer Ausgabe
      00000007          27. minimax:        ds.b 1  * 0=Mini, 1=Maxi
      00000008          28. ecma:           ds.b 1  * 1=ECMA70
      00000009          29. dichte:         ds.b 1  * 0=einfache Dichte, 1=doppelte Dichte
      0000000A          30. gaplen:         ds.b 1  * GAP 3 Laenge
      0000000B          31. spuren:         ds.b 1  * Anzahl der Spuren
      0000000C          32. kopf:           ds.b 1  * 0=einseitig, 1=zweiseitig
      0000000D          33. sektoren:       ds.b 1  * Anzahl der Sektoren pro Spur
      0000000E          34. laufwerk:       ds.b 1  * 1,2,4,8 = Laufwerkscode,
                        35.                         * 81,82,84,88 = Rueckseite
      0000000F          36. sidesel:        ds.b 1  * 0=Seite 0,SSO, $80=Seite 1 SSO
      00000010          37. sideflag:       ds.b 1  * 0=SSO nicht verwenden
      00000011          38. akttrk:         ds.b 1  * aktuelle Spur
      00000012          39. aktsek:         ds.b 1  * aktueller Sektor
                        40. *
      00000013          41. buffer:         ds.b 40 * Textbuffer fuer dez.
      0000003B          42. formtab:        ds.b 1024*7     * max bei MINI + filler+gaps
      00001C3B          43. finram:         * Ausnahme bei 8 Zoll, getrennte RAM-Controlle
                        44.
000000                  45.  section 0              * Programmspeichert
                        46.
000000                  47. anf:
000000  55AA0180        48.  dc.l $55aa0180         * Kennung fuer Bibliothek
000004  55 46 4F 52     49.  dc.b 'UFORM68K'        * Name des Programms
        4D 36 38 4B
00000C  0000099A        50.  dc.l kstart-anf        * Startadresse, relativ
000010  000016F6        51.  dc.l ende-anf          * Laenge
000014  01              52.  dc.b 1                 * relokatives Programm
000015  00 00 00        53.  dc.b 0,0,0
000018  00000000        54.  dc.l 0,0
        00000000
000020  00000000        55.  dc.l 0                 * keine weiteren Eintraege
                        56.
                        57. *** Trap-Gebiet fuer IO ***
                        58.
000024                  59. co2:            * d0.b = Zeichen
000024  48E7 0106       60.  movem.l d7/a5/a6,-(a7)
000028  3E3C 0021       61.  move #33,d7
00002C  4E41            62.  trap #1
00002E  4CDF 6080       63.  movem.l (a7)+,d7/a5/a6
000032  4E75            64.  rts
                        65.
000034                  66. ci:             * d0.b = Zeichen
000034  48E7 7FFE       67.  movem.l d1-d7/a0-a6,-(a7)
000038  3E3C 000C       68.  move #12,d7
```

```
00003C  4E41            69.  trap #1
00003E  4CDF 7FFE       70.  movem.l (a7)+,d1-d7/a0-a6
000042  4E75            71.  rts
                        72.
000044                  73. clrscreen:
000044  48E7 FFFE       74.  movem.l d0-d7/a0-a6,-(a7)
000048  3E3C 0014       75.  move #20,d7
00004C  4E41            76.  trap #1
00004E  4CDF 7FFF       77.  movem.l (a7)+,d0-d7/a0-a6
000052  4E75            78.  rts
                        79.
                        80.
000054                  81. sync:
000054  48E7 7FFE       82.  movem.l d1-d7/a0-a6,-(a7)
000058  3E3C 001C       83.  move #28,d7
00005C  4E41            84.  trap #1
00005E  4CDF 7FFE       85.  movem.l (a7)+,d1-d7/a0-a6
000062  4E75            86.  rts
                        87.
000064                  88. size:
000064  48E7 FFFE       89.  movem.l d0-d7/a0-a6,-(a7)
000068  3E3C 0019       90.  move #25,d7
00006C  4E41            91.  trap #1
00006E  4CDF 7FFF       92.  movem.l (a7)+,d0-d7/a0-a6
000072  4E75            93.  rts
                        94.
000074                  95. getram:         * a0->, A1->
000074  3E3C 0030       96.  move #59,d7
000078  4E41            97.  trap #1
00007A  4E75            98.  rts
                        99.
                       100.
00007C                 101. print4d:
00007C  48E7 FFFE      102.  movem.l d0-d7/a0-a6,-(a7)
000080  3E3C 002E      103.  move #46,d7
000084  4E41           104.  trap #1
000086  4CDF 7FFF      105.  movem.l (a7)+,d0-d7/a0-a6
00008A  4E75           106.  rts
                       107. *** Allgemeine Unterprogramme
                       108.
00008C                 109. noerr:
00008C  4240           110.  clr d0
00008E                 111. carres:
00008E  023C 00FE      112.  and.b #$fe,ccr
000092  4E75           113.  rts
                       114.
000094                 115. error:
000094  303C FFFF      116.  move #$ffff,d0
000098                 117. carset:
000098  003C 0001      118.  or.b #1,ccr
00009C  4E75           119.  rts
                       120.
00009E                 121. wai:           * Head settle time
00009E  61B4           122.  bsr sync
0000A0  67FC           123.  beq wai
0000A2                 124. wai1:
0000A2  61B0           125.  bsr sync
0000A4  67FC           126.  beq wai1      * 40 ms gesamt
0000A6  4E75           127.  rts
```

```
                        128.
                        129.
0000A8                  130. print:          * a0-> Text (achtung, pc-rel angeben)
0000A8  1018            131.  move.b (a0)+,d0
0000AA  6700 0008       132.  beq prtfin
0000AE  6100 FF74       133.  bsr co2
0000B2  60F4            134.  bra print
0000B4                  135. prtfin:
0000B4  4E75            136.  rts
                        137.
                        138. *
0000B6                  139. setup:
0000B6  102E 000E       140.  move.b laufwerk(a6),d0
0000BA  4A2E 0009       141.  tst.b dichte(a6)
0000BE  6604            142.  bne.s set1up
0000C0  803C 0010       143.  or.b #$10,d0    ; SD setzen
0000C4                  144. set1up:
0000C4  4A2E 0007       145.  tst.b miniwax(a6)
0000C8  6604            146.  bne.s set2up
0000CA  803C 0020       147.  or.b #$20,d0    ; Mini
0000CE                  148. set2up:
0000CE  11C0 FFC4       149.  move.b d0,flo4 ; Dichte wird gesetzt
0000D2  4E75            150.  rts
                        151.
0000D4                  152. ready:          * ready, dann INT
0000D4  0838 0006       153.  btst.b #6,flo4
        FFC4
0000DA  67F8            154.  beq.s ready
0000DC  1038 FFC0       155.  move.b flo0,d0 * INT loeschen und Status lesen
0000E0  4E75            156.  rts
                        157.
                        158.
0000E2                  159. rstore:
0000E2  363C 0064       160.  move #100,d3   * Anzahl der Versuche, ca. 4 sec
0000E6                  161. rist:
0000E6  1038 FFC0       162.  move.b flo0,d0 * Status loeschen
0000EA  61B2            163.  bsr wai        * sicherheitshalber warten 40ms
0000EC  61C8            164.  bsr setup      *  Parameter definieren
0000EE  11FC 0008       165.  move.b #$b,flo0 * langsamste Steprate nehmen
        FFC0
0000F4  61DE            166.  bsr ready      * warten bis INT
0000F6  0800 0007       167.  btst #7,d0
0000FA  6792            168.  beq carres     * fertig, kein Fehler
0000FC  51C8 FFE8       169.  dbra d3,rist   * sonst neuer Versuch
000100  41FA 0BB5       170.  lea rmsg(pc),a0
000104  6000 0870       171.  bra wstart
                        172.
                        173.
000108                  174. stepin:         * Step ausfuehren
000108  6194            175.  bsr wai
00010A  61AA            176.  bsr setup
00010C  11FC 0058       177.  move.b #$5b,flo0       * Step ohne Verify
        FFC0
000112  61C0            178.  bsr ready      * bis ausgefuehrt
000114  4E75            179.  rts            * Ende
                        180.
000116                  181. read:           * Lesen eines Sektors
000116  619E            182.  bsr setup      * vorbereiten
000118  11EE 0012       183.  move.b aktsek(a6),flo2 * Sektor einstellen
        FFC2
```

```
00011E  41EE 003B       184.  lea formtab(a6),a0     * Zieladresse
000122  43F8 FFC0       185.  lea flo0,a1     * Befehl
000126  45F8 FFC4       186.  lea flo4,a2     * Status
00012A  47F8 FFC3       187.  lea flo3,a3     * Daten
00012E  102E 000F       188.  move.b sidesel(a6),d0
000132  C03C 0080       189.  and.b #$80,d0  * Seiten Controlle
000136  E518            190.  rol.b #2,d0
000138  C03C 0002       191.  and.b #%00000010,d0
00013C  D03C 0088       192.  add.b #$88,d0  * head schon down
000140  1280            193.  move.b d0,(a1) * und Starten
000142                  194. rd1:
000142  1012            195.  move.b (a2),d0 * Status pruefen
000144  E318            196.  rol.b #1,d0     * setzen der Flags
000146  6B06            197.  bmi.s rd1e      * INT
000148  64F8            198.  bcc.s rd1       * DRQ
00014A  10D3            199.  move.b (a3),(a0)+      * und laden
00014C  60F4            200.  bra.s rd1       * immer weiter
00014E                  201. rd1e:
00014E  1011            202.  move.b (a1),d0 * Status einlesen
000150  4E75            203.  rts             * Status in D0.B
                        204.
                        205.
000152                  206. form:            * Formatieren einer Spur
000152  6100 FF4A       207.  bsr wai         * erst mal Warten fuer Head-Settle
000156  6100 FF5E       208.  bsr setup
00015A  41EE 003B       209.  lea formtab(a6),a0     * Startadresse Datenquelle
00015E  43F8 FFC0       210.  lea flo0,a1     * Befehl
000162  45F8 FFC4       211.  lea flo4,a2     * Status
000166  47F8 FFC3       212.  lea flo3,a3     * Daten
00016A  102E 000F       213.  move.b sidesel(a6),d0
00016E  C03C 0080       214.  and.b #$80,d0  * Seiten Controlle
000172  E518            215.  rol.b #2,d0
000174  C03C 0002       216.  and.b #%00000010,d0
000178  D03C 00F4       217.  add.b #$f4,d0  * Formatierbefehl ausgeben
00017C  1280            218.  move.b d0,(a1) * Befehl ausgeben
00017E                  219. fmtlp:
00017E  1012            220.  move.b (a2),d0 * Status pruefen
000180  E318            221.  rol.b #1,d0
000182  6B06            222.  bmi.s fmte      * Ende erreiche
000184  64F8            223.  bcc.s fmtlp     * sonst auf DRQ warten
000186  1698            224.  move.b (a0)+,(a3) * und ausgeben
000188  60F4            225.  bra.s fmtlp
00018A                  226. fmte:
00018A  1011            227.  move.b (a1),d0 * INT loeschen, Status lesen
00018C  C03C 0044       228.  and.b #$44,d0
000190  4E75            229.  rts             * Schreibschutz und Datalost error
                        230.
                        231. *
000192                  232. format:                 * Formatierung ausfuehren
000192  422E 0006       233.  clr.b chars(a6)       * Zeichenzaehler auf 0
000196  41FA 0BEA       234.  lea fmsg(pc),a0
00019A  6100 FF0C       235.  bsr print
00019E  1038 FFC0       236.  move.b flo0,d0        * Status loeschen
0001A2  6100 FF3E       237.  bsr rstore            * erst mal auf Spur 0 bringen
0001A6  422E 0011       238.  clr.b akttrk(a6)      * Start bei Spur 0
0001AA  162E 000B       239.  move.b spuren(a6),d3  * Schleifenzaehler
0001AE                  240. loop:
0001AE  3F03            241.  move d3,-(a7)
0001B0  6100 00FC       242.  bsr ecmachk
```

```
0001B4  6100 01D4       243.  bsr gentrack
0001B8  6198            244.  bsr form
0001BA  6100 0168       245.  bsr errchk
0001BE  102E 000C       246.  move.b kopf(a6),d0
0001C2  B03C 0001       247.  if.b d0 <eq> #1 then.s
        6628
0001C8  002E 0080       248.     or.b #$80,laufwerk(a6) * zweite Seite auch
        000E
0001CE  4A2E 0010       249.     tst.b sideflag(a6)
0001D2  6706            250.     if <ne> then.s
0001D4  1D7C 0080       251.        move.b #$80,sidesel(a6)
        000F
                        252.     endi
0001DA  6100 01AE       253.     bsr gentrack
0001DE  6100 FF72       254.     bsr form
0001E2  6100 0140       255.     bsr errchk
0001E6  022E 007F       256.     and.b #$7f,laufwerk(a6)
        000E
0001EC  422E 000F       257.     clr.b sidesel(a6)
                        258.  endi
0001F0  522C 0011       259.  add.b #1,akttrk(a6)
0001F4  102E 0011       260.  move.b akttrk(a6),d0
0001F8  B02C 000B       261.  cmp.b spuren(a6),d0
0001FC  6704            262.  if <ne> then.s
0001FE  6100 FF08       263.     bsr stepin    * letzte Spur nicht mehr
                        264.  endi
000202  102E 0006       265.  move.b chars(a6),d0
000206  B03C 0028       266.  if.b d0 <eq> #40 then.s
        6608
00020C  422E 0006       267.     clr.b chars(a6)
000210  6100 010A       268.     bsr crlf
                        269.  endi
000214  522E 0006       270.  add.b #1,chars(a6)
000218  103C 0046       271.  move.b #'F',d0
00021C  6100 FE06       272.  bsr co2         * ausgeben als Kontrolle
000220  361F            273.  move (a7)+,d3
000222  5303            274.  sub.b #1,d3
000224  6688            275.  bne loop
                        276. * dann Prueflesen
000226                  277. pruefen:
000226  422E 0006       278.  clr.b chars(a6)
00022A  41FA 0D5D       279.  lea vmsg(pc),a0
00022E  6100 FE78       280.  bsr print
000232  6100 FEAE       281.  bsr rstore
000236  422E 0011       282.  clr.b akttrk(a6)
00023A  162E 000B       283.  move.b spuren(a6),d3   * Schleifenzaehler
00023E                  284. loop1:
00023E  3F03            285.  move d3,-(a7)
000240  6100 006C       286.  bsr ecmachk
000244  6100 02AC       287.  bsr verify
000248  102E 000C       288.  move.b kopf(a6),d0
00024C  B03C 0001       289.  if.b d0 <eq> #1 then.s
        6620
000252  002E 0080       290.     or.b #$80,laufwerk(a6) * zweite Seite auch
        000E
000258  4A2E 0010       291.     tst.b sideflag(a6)
00025C  6706            292.     if <ne> then.s
00025E  1D7C 0080       293.        move.b #$80,sidesel(a6)
        000F
```

```
                                294.      endi
000264  6100 028C               295.      bsr verify
000268  022E 007F               296.      and.b #$7f,laufwerk(a6)
        000E
00026E  422E 000F               297.      clr.b sidesel(a6)
                                298.   endi
000272  522E 0011               299.   add.b #1,akttrk(a6)
000276  102E 0011               300.   move.b akttrk(a6),d0
00027A  B02E 000B               301.   cmp.b spuren(a6),d0
00027E  6704                    302.   if <ne> then.s
000280  6100 FE86               303.      bsr stepin     * letzte Spur nicht mehr
                                304.   endi
000284  102E 0006               305.   move.b chars(a6),d0
000288  B03C 0028               306.   if.b d0 <eq> #40 then.s
        6608
00028E  422E 0006               307.      clr.b chars(a6)
000292  6100 0088               308.      bsr crlf
                                309.   endi
000296  522E 0006               310.   add.b #1,chars(a6)
00029A  103C 0056               311.   move.b #'V',d0
00029E  6100 FD84               312.   bsr co2          * ausgeben als Kontrolle
0002A2  361F                    313.   move (a7)+,d3
0002A4  5303                    314.   sub.b #1,d3
0002A6  6696                    315.   bne loop1
0002A8  6100 FE38               316.   bsr rstore      * und am Schluss zurueck
0002AC  4E75                    317.   rts
                                318.
0002AE                          319. ecmachk:          * fuer ECMA 70 Format
0002AE  4A2E 0008               320.   tst.b ecma(a6)
0002B2  6700 FDDA               321.   beq carres      * Ende, kein ECMA-Format
0002B6  422E 0007               322.   clr.b minimax(a6)       * immer Minilaufwerk
0002BA  102E 0011               323.   move.b akttrk(a6),d0    * aktuelle Spur
0002BE  B03C 0000               324.   if.b d0 <eq> #0 then.s
        6622
0002C4  3D7C 0080               325.      move.w #128,anzbyte(a6)
        0004
0002CA  422E 0009               326.      clr.b dichte(a6)
0002CE  422E 000C               327.      clr.b kopf(a6)
0002D2  1D7C 0010               328.      move.b #16,sektoren(a6)
        000D
0002D8  1D7C 001B               329.      move.b #27,gaplen(a6)
        000A
0002DE  1D7C 0028               330.      move.b #40,spuren(a6)
        000B
0002E4  6022                    331.   else.s
0002E6  3D7C 0100               332.      move #256,anzbyte(a6)
        0004
0002EC  1D7C 0001               333.      move.b #1,dichte(a6)
        0009
0002F2  422E 000C               334.      clr.b kopf(a6)
0002F6  1D7C 0010               335.      move.b #16,sektoren(a6)
        000D
0002FC  1D7C 0036               336.      move.b #54,gaplen(a6)
        000A
000302  1D7C 0028               337.      move.b #40,spuren(a6)
        000B
                                338.   endi
000308  4E75                    339.   rts
                                340.
```

```
00030A                  341. printdez:       * Ausgabe dezimal, Wert in d0
00030A  41EE 0013       342.  lea buffer(a6),a0
00030E  6100 FD6C       343.  bsr print4d    * in Buffer ausgeben
000312  41EE 0013       344.  lea buffer(a6),a0
000316  6100 FD90       345.  bsr print      * dann auf die Konsole
00031A  4E75            346.  rts
                        347.
                        348.
00031C                  349. crlf:
00031C  41FA 0C68       350.  lea tcrlf(pc),a0
000320  6000 FD86       351.  bra print
                        352.
000324                  353. errchk:
000324  4A00            354.  tst.b d0
000326  6700 FD66       355.  beq carres     * =0 , keine Fehler da
00032A  3F00            356.  move d0,-(a7)
00032C  41FA 0C40       357.  lea tkmsg(pc),a0
000330  6100 FD76       358.  bsr print
000334  4240            359.  clr d0
000336  102E 0011       360.  move.b akttrk(a6),d0
00033A  61CE            361.  bsr printdez
00033C  41FA 0C3D       362.  lea skmsg(pc),a0
000340  6100 FD66       363.  bsr print
000344  4240            364.  clr d0
000346  102E 0012       365.  move.b aktsek(a6),d0
00034A  61BE            366.  bsr printdez
00034C  61CE            367.  bsr crlf
00034E  301F            368.  move (a7)+,d0
000350  41FA 0C05       369.  lea f0msg(pc),a0
000354  0800 0002       370.  btst #2,d0
000358  6704            371.  if <ne> then.s
00035A  41FA 0B7B       372.     lea f2msg(pc),a0
                        373.  endi
00035E  0800 0003       374.  btst #3,d0
000362  6704            375.  if <ne> then.s
000364  41FA 0B4D       376.     lea f3msg(pc),a0
                        377.  endi
000368  0800 0004       378.  btst #4,d0
00036C  6704            379.  if <ne> then.s
00036E  41FA 0B20       380.     lea f4msg(pc),a0
                        381.  endi
000372  0800 0005       382.  btst #5,d0
000376  6704            383.  if <ne> then.s
000378  41FA 0AF3       384.     lea f5msg(pc),a0
                        385.  endi
00037C  0800 0006       386.  btst #6,d0
000380  6704            387.  if <ne> then.s
000382  41FA 0A2D       388.     lea f6msg(pc),a0
                        389.  endi
000386  6000 05EE       390.  bra wstart     * wieder zurueck mit Fehlerausgabe
                        391.
00038A                  392. gentrack:
00038A  41EE 0038       393.  lea formtab(a6),a0     * Dorthin Formatierdaten legen
00038E  4A2E 0009       394.  tst.b dichte(a6)
000392  660A            395.  if <eq> then.s * SD
000394  323C 000F       396.     move #16-1,d1 * Anzahl
000398  143C 00FF       397.     move.b #$ff,d2 * Fuellbyte
00039C  6008            398.  else.s          * DD
00039E  323C 001F       399.     move #32-1,d1
```

```
0003A2  143C 004E       400.      move.b #$4e,d2
                        401.   endi
0003A6                  402. gen1trk:
0003A6  10C2            403.   move.b d2,(a0)+
0003A8  51C9 FFFC       404.   dbra d1,gen1trk
0003AC  1D7C 0001       405.   move.b #1,aktsek(a6)   * Startsektor
        0012
0003B2  122E 000D       406.   move.b sektoren(a6),d1
0003B6                  407. gen2trk:        * Sektorschleife
0003B6  3F01            408.   move d1,-(a7)
0003B8  4A2E 0009       409.   tst.b dichte(a6)
0003BC  6600 0062       410.   if <eq> then   * SD
0003C0  323C 0005       411.      move #6-1,d1
0003C4                  412.    gen3trk:
0003C4  4218            413.      clr.b (a0)+
0003C6  51C9 FFFC       414.      dbra d1,gen3trk
0003CA  10FC 00FE       415.      move.b #$fe,(a0)+        * ID-Adr-Mark
0003CE  10EE 0011       416.      move.b akttrk(a6),(a0)+  * Spur
0003D2  102E 000F       417.      move.b sidesel(a6),d0
0003D6  C03C 0080       418.      and.b #$80,d0
0003DA  6606            419.      if <eq> then.s
0003DC  10FC 0000       420.         move.b #0,(a0)+         * Seite 0
0003E0  6004            421.      else.s
0003E2  10FC 0001       422.         move.b #1,(a0)+         * Seite 1
                        423.      endi
0003E6  10EE 0012       424.      move.b aktsek(a6),(a0)+  * Sektor
0003EA  302E 0004       425.      move.w anzbyte(a6),d0    * 128,256,512,1024
0003EE  E058            426.      ror #8,d0
0003F0  B03C 0004       427.      if.b d0 <eq> #4 then.s   * 0,  1,  2,  4,
        6604
0003F6  103C 0003       428.         move.b #3,d0
                        429.      endi
0003FA  10C0            430.      move.b d0,(a0)+          * Laengenkennung
0003FC  10FC 00F7       431.      move.b #$f7,(a0)+        * CRC erzeugen
000400  323C 000A       432.      move #11-1,d1
000404                  433.    gen4trk:
000404  10FC 00FF       434.      move.b #$ff,(a0)+
000408  51C9 FFFA       435.      dbra d1,gen4trk
00040C  323C 0005       436.      move #6-1,d1
000410                  437.    gen5trk:
000410  10FC 0000       438.      move.b #$0,(a0)+
000414  51C9 FFFA       439.      dbra d1,gen5trk
000418  10FC 00FB       440.      move.b #$fb,(a0)+        * DATA Mark
00041C  6000 0076       441.   else           * DD
000420  323C 000B       442.      move #12-1,d1
000424                  443.    gen3atrk:
000424  4218            444.      clr.b (a0)+
000426  51C9 FFFC       445.      dbra d1,gen3atrk
00042A  323C 0002       446.      move #3-1,d1
00042E                  447.    gen3btrk:
00042E  10FC 00F5       448.      move.b #$f5,(a0)+
000432  51C9 FFFA       449.      dbra d1,gen3btrk
000436  10FC 00FE       450.      move.b #$fe,(a0)+        * ID-Adr-Mark
00043A  10EE 0011       451.      move.b akttrk(a6),(a0)+  * Spur
00043E  102E 000F       452.      move.b sidesel(a6),d0
000442  C03C 0080       453.      and.b #$80,d0
000446  6606            454.      if <eq> then.s
000448  10FC 0000       455.         move.b #0,(a0)+         * Seite 0
00044C  6004            456.      else.s
```

```
00044E  10FC 0001       457.        move.b #1,(a0)+         * Seite 1
                        458.      endi
000452  10EE 0012       459.      move.b aktsek(a6),(a0)+  * Sektor
000456  302E 0004       460.      move.w anzbyte(a6),d0    * 128,256,512,1024
00045A  E058            461.      ror #8,d0
00045C  B03C 0004       462.      if.b d0 <eq> #4 then.s   * 0,  1,  2,  4,
        6604
000462  103C 0003       463.         move.b #3,d0
                        464.      endi
000466  10C0            465.      move.b d0,(a0)+          * Laengenkennung
000468  10FC 00F7       466.      move.b #$f7,(a0)+        * CRC erzeugen
00046C  323C 0015       467.      move #22-1,d1
000470                  468.    gen4atrk:
000470  10FC 004E       469.      move.b #$4e,(a0)+
000474  51C9 FFFA       470.      dbra d1,gen4atrk
000478  323C 0008       471.      move #12-1,d1
00047C                  472.    gen5atrk:
00047C  10FC 0000       473.      move.b #$0,(a0)+
000480  51C9 FFFA       474.      dbra d1,gen5atrk
000484  323C 0002       475.      move #3-1,d1
000488                  476.    gen6atrk:
000488  10FC 00F5       477.      move.b #$f5,(a0)+
00048C  51C9 FFFA       478.      dbra d1,gen6atrk
000490  10FC 00FB       479.      move.b #$fb,(a0)+        * DATA Mark
                        480.    endi
                        481.  * Datenblock ablegen
000494  322E 0004       482.    move anzbyte(a6),d1
000498  5341            483.    sub #1,d1                * fuer DBRA
00049A                  484.  gen7btrk:
00049A  10FC 00E5       485.    move.b #$e5,(a0)+        * Filler Wert
00049E  51C9 FFFA       486.    dbra d1,gen7btrk
0004A2  10FC 00F7       487.    move.b #$f7,(a0)+        * und CRC danach
0004A6  4A2E 0009       488.    tst.b dichte(a6)
0004AA  6606            489.    if <eq> then.s
0004AC  103C 00FF       490.       move.b #$ff,d0          * Block Gap 3
0004B0  6004            491.    else.s
0004B2  103C 004E       492.       move.b #$4e,d0
                        493.    endi
0004B6  4241            494.    clr d1
0004B8  122E 000A       495.    move.b gaplen(a6),d1
0004BC  5341            496.    sub #1,d1
0004BE                  497.  gen7trk:
0004BE  10C0            498.    move.b d0,(a0)+
0004C0  51C9 FFFC       499.    dbra d1,gen7trk
0004C4  522E 0012       500.    add.b #1,aktsek(a6)      * naechster Sektor dann
0004C8  321F            501.    move (a7)+,d1
0004CA  5301            502.    sub.b #1,d1
0004CC  6600 FEE8       503.    bne gen2trk     * bis alle Sektoren ausgegeben sind
                        504.  *
0004D0  4A2E 0009       505.    tst.b dichte(a6)
0004D4  6606            506.    if <eq> then.s
0004D6  103C 00FF       507.       move.b #$ff,d0          * Block Gap 3
0004DA  6004            508.    else.s
0004DC  103C 004E       509.       move.b #$4e,d0
                        510.    endi            * Rest fuellen
0004E0  323C 03E7       511.    move #1000-1,d1 * Sicherheitsfaktor
0004E4                  512.  gap8trk:
0004E4  10C0            513.    move.b d0,(a0)+
0004E6  B1C8            514.    cmpa.l a0,a7    * a0 muss kleiner sein als A7
```

```
0004E8  6500 0474       515.  bcs ramerror    * sonst RAM-Fehler, zuwenig RAM
0004EC  51C9 FFF6       516.  dbra d1,gap8trk * Stack unterlauf
0004F0  4E75            517.  rts            * Ende Format Aufbereitung
                        518.
0004F2                  519. verify:         * pruefen einer Spur
0004F2  6100 FBAA       520.  bsr wai
0004F6  1D7C 0001       521.  move.b #1,aktsek(a6)
        0012
0004FC  122E 000D       522.  move.b sektoren(a6),d1
000500                  523. verlp:
000500  3F01            524.  move d1, (a7)
000502  6100 FC12       525.  bsr read       * immer Lesen
000506  6100 FE1C       526.  bsr errchk     * und pruefen
00050A  522E 0012       527.  add.b #1,aktsek(a6)
00050E  321F            528.  move (a7)+,d1
000510  5301            529.  sub.b #1,d1
000512  66EC            530.  bne verlp
000514  4E75            531.  rts
                        532.
                        533.
                        534. *** Menues
                        535.
                        536.
000516                  537. setspur:
000516  41FA 0C17       538.  lea menue3(pc),a0
00051A  6100 F88C       539.  bsr print
00051E  6100 FB14       540.  bsr ci
000522  B03C 0031       541.  if.b d0 <eq> #'1' then.s
        6606
000528  103C 0023       542.     move.b #35,d0
00052C  6032 B03C       543.  else.s if.b d0 <eq> #'2' then.s
        0032 6606
000534  103C 0028       544.     move.b #40,d0
000538  6026 B03C       545.  else.s if.b d0 <eq> #'3' then.s
        0033 6606
000540  103C 0046       546.     move.b #70,d0
000544  601A B03C       547.  else.s if.b d0 <eq> #'4' then.s
        0034 6606
00054C  103C 004D       548.     move.b #77,d0
000550  600E B03C       549.  else.s if.b d0 <eq> #'5' then.s
        0035 6606
000558  103C 0050       550.     move.b #80,d0
00055C  6002            551.  else.s
00055E  60B6            552.     bra setspur
                        553.  endi
000560  1D40 000B       554.  move.b d0,spuren(a6)
000564  4E75            555.  rts
                        556.
                        557.
000566                  558. setlw:
000566  41FA 0C18       559.  lea menue4(pc),a0
00056A  6100 FB3C       560.  bsr print
00056E  6100 FAC4       561.  bsr ci
000572  B03C 0031       562.  if.b d0 <eq> #'1' then.s
        6606
000578  422E 000C       563.     clr.b kopf(a6)
00057C  6024 B03C       564.  else.s if.b d0 <eq> #'2' then.s
        0032 6608
000584  1D7C 0001       565.     move.b #1,kopf(a6)
        000C
```

```
00058A  6016 B03C       566.  else.s if.b d0 <eq> #'3' then.s
        0033 660E
000592  1D7C 0001       567.     move.b #1,kopf(a6)
        000C
000598  1D7C 0001       568.     move.b #1,sideflag(a6)
        0010
00059E  6002            569.  else.s
0005A0  60C4            570.     bra setl~
                        571.  endi
0005A2  4E75            572.  rts
                        573.
0005A4                  574. setdichte:
0005A4  41FA 0B4B       575.  lea menue2(pc),a0
0005A8  6100 FAFE       576.  bsr print
0005AC  6100 FA86       577.  bsr ci
000580  B03C 0031       578.  if.b d0 <eq> #'1' then.s
        6606
0005B6  103C 0000       579.     move.b #0,d0  * einfache Dichte
0005BA  600E B03C       580.  else.s if.b d0 <eq> #'2' then.s
        0032 6606
0005C2  103C 0001       581.     move.b #1,d0  * doppelte Dichte
0005C6  6002            582.  else.s
0005C8  60DA            583.     bra setdichte
                        584.  endi
0005CA  1D40 0009       585.  move.b d0,dichte(a6)
0005CE  4E75            586.  rts
                        587.
0005D0                  588. setsek:
0005D0  102E 0007       589.  move.b minimax(a6),d0
0005D4  B03C 0000       590.  if.b d0 <eq> #0 then   * MINI
        6600 00E8
0005DC  102E 0009       591.     move.b dichte(a6),d0
0005E0  B03C 0000       592.     if.b d0 <eq> #0 then  * SD
        6600 0062
0005E8  41FA 0BF6       593.        lea menue5a(pc),a0
0005EC  6100 FABA       594.        bsr print
0005F0                  595.     set1sek:
0005F0  6100 FA42       596.        bsr ci
0005F4  B03C 0031       597.        if.b d0 <eq> #'1' then.s
        6614
0005FA  3D7C 0080       598.           move.w #128,anzbyte(a6)
        0004
000600  1D7C 0010       599.           move.b #16,sektoren(a6)
        000D
000606  1D7C 0018       600.           move.b #27,gaplen(a6)
        000A
00060C  6036 B03C       601.        else.s if.b d0 <eq> #'2' then.s
        0032 6614
000614  3D7C 0080       602.           move.w #128,anzbyte(a6)
        0004
00061A  1D7C 0012       603.           move.b #18,sektoren(a6)
        000D
000620  1D7C 000A       604.           move.b #10,gaplen(a6)
        000A
000626  601C B03C       605.        else.s if.b d0 <eq> #'3' then.s
        0033 6614
00062E  3D7C 0100       606.           move.w #256,anzbyte(a6)
        0004
000634  1D7C 000A       607.           move.b #10,sektoren(a6)
        000D
```

```
00063A  1D7C 0010       608.            move.b #16,gaplen(a6)
        000A
000640  6002            609.         else.s
000642  60AC            610.            bra set1sek
                        611.         endi
000644  6000 0078       612.      else                    x DD
000648  41FA 0C17       613.         lea menue5b(pc),a0
00064C  6100 FA5A       614.         bsr print
000650                  615.      set2sek:
000650  6100 F9E2       616.         bsr ci
000654  B03C 0031       617.         if.b d0 <eq> #'1' then.s
        6614
00065A  3D7C 0100       618.            move.w #256,anzbyte(a6)
        0004
000660  1D7C 0010       619.            move.b #16,sektoren(a6)
        000D
000666  1D7C 0036       620.            move.b #54,gaplen(a6)
        000A
00066C  6050 B03C       621.         else.s if.b d0 <eq> #'2' then.s
        0032 6614
000674  3D7C 0200       622.            move.w #512,anzbyte(a6)
        0004
00067A  1D7C 0009       623.            move.b #9,sektoren(a6)
        000D
000680  1D7C 0036       624.            move.b #54,gaplen(a6)
        000A
000686  6036 B03C       625.         else.s if.b d0 <eq> #'3' then.s
        0033 6614
00068E  3D7C 0200       626.            move.w #512,anzbyte(a6)
        0004
000694  1D7C 000A       627.            move.b #10,sektoren(a6)
        000D
00069A  1D7C 0028       628.            move.b #40,gaplen(a6)
        000A
0006A0  601C B03C       629.         else.s if.b d0 <eq> #'4' then.s
        0034 6614
0006A8  3D7C 0400       630.            move.w #1024,anzbyte(a6)
        0004
0006AE  1D7C 0005       631.            move.b #5,sektoren(a6)
        000D
0006B4  1D7C 0036       632.            move.b #54,gaplen(a6)
        000A
0006BA  6002            633.         else.s
0006BC  6092            634.            bra set2sek
                        635.         endi
                        636.      endi
0006BE  6000 010A       637.   else                      x MAXI
0006C2  102E 0009       638.      move.b dichte(a6),d0
0006C6  B03C 0000       639.      if.b d0 <eq> #0 then  x SD
        6600 0048
0006CE  41FA 0C3B       640.         lea menue5c(pc),a0
0006D2  6100 F9D4       641.         bsr print
0006D6                  642.      set3sek:
0006D6  6100 F95C       643.         bsr ci
0006DA  B03C 0031       644.         if.b d0 <eq> #'1' then.s
        6614
0006E0  3D7C 0080       645.            move.w #128,anzbyte(a6)
        0004
0006E6  1D7C 001A       646.            move.b #26,sektoren(a6)
        000D
```

```
0006EC  1D7C 001B        647.          move.b #27,gaplen(a6)
        000A
0006F2  601C B03C        648.       else.s if.b d0 <eq> #'2' then.s
        0032 6614
0006FA  3D7C 0100        649.          move.w #256,anzbyte(a6)
        0004
000700  1D7C 000F        650.          move.b #15,sektoren(a6)
        000D
000706  1D7C 001B        651.          move.b #27,gaplen(a6)
        000A
00070C  6002             652.       else.s
00070E  60C6             653.          bra set3sek
                         654.       endi
000710  6000 00B8        655.    else                    * DD
000714  41FA 0C4D        656.       lea menue5d(pc),a0
000718  6100 F98E        657.       bsr print
00071C                   658.    set4sek:
00071C  6100 F916        659.       bsr ci
000720  B03C 0031        660.       if.b d0 <eq> #'1' then
        6600 0018
000728  3D7C 0100        661.          move.w #256,anzbyte(a6)
        0004
00072E  1D7C 001A        662.          move.b #26,sektoren(a6)
        000D
000734  1D7C 0036        663.          move.b #54,gaplen(a6)
        000A
00073A  6000 008E        664.       else if.b d0 <eq> #'2' then
        B03C 0032 6600
        0018
000746  3D7C 0200        665.          move.w #512,anzbyte(a6)
        0004
00074C  1D7C 000E        666.          move.b #14,sektoren(a6)
        000D
000752  1D7C 0036        667.          move.b #54,gaplen(a6)
        000A
000758  6000 0070        668.       else if.b d0 <eq> #'3' then
        B03C 0033 6600
        0016
000764  3D7C 0200        669.          move.w #512,anzbyte(a6)
        0004
00076A  1D7C 000F        670.          move.b #15,sektoren(a6)
        000D
000770  1D7C 0036        671.          move.b #54,gaplen(a6)
        000A
000776  6052 B03C        672.       else.s if.b d0 <eq> #'4' then.s
        0034 6614
00077E  3D7C 0200        673.          move.w #512,anzbyte(a6)
        0004
000784  1D7C 0010        674.          move.b #16,sektoren(a6)
        000D
00078A  1D7C 0028        675.          move.b #40,gaplen(a6)
        000A
000790  6038 B03C        676.       else.s if.b d0 <eq> #'5' then.s
        0035 6614
000798  3D7C 0400        677.          move.w #1024,anzbyte(a6)
        0004
00079E  1D7C 0008        678.          move.b #8,sektoren(a6)
        000D
0007A4  1D7C 0036        679.          move.b #54,gaplen(a6)
        000A
```

```
0007AA  601E B03C       680.        else.s if.b d0 <eq> #'6' then.s
        0036 6614
0007B2  3D7C 0400       681.           move.w #1024,anzbyte(a6)
        0004
0007B8  1D7C 0009       682.           move.b #9,sektoren(a6)
        000D
0007BE  1D7C 0028       683.           move.b #40,gaplen(a6)
        000A
0007C4  6004            684.        else.s
0007C6  6000 FF54       685.           bra set4sek
                        686.        endi
                        687.     endi
                        688.  endi
0007CA  4E75            689.  rts
                        690.
                        691.
0007CC                  692. werteaus:
0007CC  103C 001A       693.  move.b #$1a,d0
0007D0  6100 F852       694.  bsr co2                  * Bildschirm loeschen
0007D4  4A2E 0007       695.  tst.b minimax(a6)
0007D8  6606            696.  if <eq> then.s
0007DA  41FA 0DE8       697.     lea tmini(pc),a0
0007DE  6004            698.  else.s
0007E0  41FA 0DF2       699.     lea tmaxi(pc),a0
                        700.  endi
0007E4  6100 F8C2       701.  bsr print
0007E8  4A2E 0009       702.  tst.b dichte(a6)
0007EC  6606            703.  if <eq> then.s
0007EE  41FA 0DF4       704.     lea tfm(pc),a0
0007F2  6004            705.  else.s
0007F4  41FA 0DF3       706.     lea tmfm(pc),a0
                        707.  endi
0007F8  102E 0008       708.  move.b ecma(a6),d0
0007FC  B03C 0001       709.  if.b d0 <eq> #1 then.s
        6604
000802  41FA 0DE8       710.     lea tecma(pc),a0
                        711.  endi
000806  6100 F8A0       712.  bsr print
00080A  6100 FB10       713.  bsr crlf
00080E  6100 FB0C       714.  bsr crlf
000812  41FA 0DE3       715.  lea tspuren(pc),a0
000816  6100 F890       716.  bsr print
00081A  4240            717.  clr d0
00081C  102E 0008       718.  move.b spuren(a6),d0
000820  6100 FAE8       719.  bsr printdez
000824  6100 FAF6       720.  bsr crlf
000828  41FA 0DE1       721.  lea tsektoren(pc),a0
00082C  6100 F87A       722.  bsr print
000830  4240            723.  clr d0
000832  102E 000D       724.  move.b sektoren(a6),d0
000836  6100 FAD2       725.  bsr printdez
00083A  6100 FAE0       726.  bsr crlf
00083E  41FA 0DDF       727.  lea tbytes(pc),a0
000842  6100 F864       728.  bsr print
000846  302E 0004       729.  move.w anzbyte(a6),d0
00084A  6100 FABE       730.  bsr printdez
00084E  6100 FACC       731.  bsr crlf
000852  41FA 0DDF       732.  lea tlaufwerk(pc),a0
000856  6100 F850       733.  bsr print
```

```
00085A  102E 000E       734.  move.b laufwerk(a6),d0
00085E  C03C 000F       735.  and.b #$f,d0
000862  B03C 0001       736.  if.b d0 <eq> #1 then.s
        6606
000868  103C 0041       737.     move.b #'A',d0
00086C  602A B03C       738.  else.s if.b d0 <eq> #2 then.s
        0002 6606
000874  103C 0042       739.     move.b #'B',d0
000878  601E B03C       740.  else.s if.b d0 <eq> #4 then.s
        0004 6606
000880  103C 0043       741.     move.b #'C',d0
000884  6012 B03C       742.  else.s if.b d0 <eq> #8 then.s
        0008 6608
00088C  103C 0044       743.     move.b #'D',d0
000890  6000 0006       744.  else
000894  103C 003F       745.     move.b #'?',d0
                        746.  endi
000898  6100 F78A       747.  bsr co2
00089C  6100 FA7E       748.  bsr crlf
0008A0  6100 FA7A       749.  bsr crlf
0008A4  102E 000C       750.  move.b kopf(a6),d0
0008A8  B03C 0001       751.  if.b d0 <eq> #1 then   * zweiseitig
        6600 001A
0008B0  4A2E 0010       752.     tst.b sideflag(a6)
0008B4  6606            753.     if <eq> then.s
0008B6  41FA 0DDF       754.        lea tzweiseitig(pc),a0
0008BA  6004            755.     else.s
0008BC  41FA 0DF8       756.        lea tzweisso(pc),a0
                        757.     endi
0008C0  6100 F7E6       758.     bsr print
0008C4  6000 002A       759.  else                    * einseitig
0008C8  102E 000E       760.     move.b laufwerk(a6),d0
0008CC  C03C 0080       761.     and.b #$80,d0
0008D0  660A            762.     if <eq> then.s
0008D2  41FA 0D69       763.        lea teinseitig(pc),a0
0008D6  6100 F7D0       764.        bsr print
0008DA  6014            765.     else.s
0008DC  4A2E 0010       766.        tst.b sideflag(a6)
0008E0  6606            767.        if <eq> then.s
0008E2  41FA 0D75       768.           lea trueckseite(pc),a0
0008E6  6004            769.        else.s
0008E8  41EE'1674       770.           lea truecksso(a6),a0
                        771.        endi
0008EC  6100 F7BA       772.        bsr print
                        773.     endi
                        774.  endi
0008F0  6100 FA2A       775.  bsr crlf
0008F4  6100 FA26       776.  bsr crlf       * Kapazitaet ausrechnen
0008F8  41FA 0CAB       777.  lea tkap(pc),a0
0008FC  6100 F7AA       778.  bsr print
000900  4240            779.  clr d0
000902  4241            780.  clr d1
000904  102E 000D       781.  move.b sektoren(a6),d0
000908  122E 000B       782.  move.b spuren(a6),d1
00090C  C0C1            783.  mulu d1,d0      * spuren*sektoren
00090E  122E 000C       784.  move.b kopf(a6),d1
000912  B23C 0001       785.  if.b d1 <eq> #1 then.s
        6602
000918  D080            786.     add.l d0,d0   * wenn zweiseitig, dann mal 2
```

```
                        787.  endi
00091A  322E 0004       788.  move anzbyte(a6),d1
00091E  B27C 0080       789.  if.w d1 <eq> #128 then.s
        6604
000924  E640           790.     asr #3,d0      * div 8 in K
000926  6024 B27C       791.  else.s if.w d1 <eq> #256 then.s
        0100 6604
00092E  E440           792.     asr #2,d0     * div 4 in K
000930  601A B27C       793.  else.s if.w d1 <eq> #512 then.s
        0200 6604
000938  E240           794.     asr #1,d0     * div 2 in K
00093A  6010 B27C       795.  else.s if.w d1 <eq> #1024 then.s
        0400 6606
000942  4E71           796.     nop            * ok.
000944  6000 0006       797.  else
000948  303C 0000       798.     move #0,d0     * FEHLER
                        799.  endi
00094C  6100 F9BC       800.  bsr printdez
000950  41FA 0C6A       801.  lea tkap1(pc),a0
000954  6100 F752       802.  bsr print
000958  6100 F9C2       803.  bsr crlf
00095C  4E75            804.  rts
                        805.
                        806. *** Hauptprogramm
                        807.
00095E                  808. ramerror:        * Fehler zuwenig RAM, abbrechen
00095E  303C 0021       809.  move #$21,d0    * Grosse Schrift
000962  6100 F700       810.  bsr size
000966  6100 F6DC       811.  bsr clrscreen   * und Bildschirm loeschen
00096A  41FA 0290       812.  lea txterr(pc),a0
00096E  6100 F738       813.  bsr print
000972                  814. ramwa:
000972  4E71            815.  nop             * Sicherheitshalber
000974  60FC            816.  bra ramwa
                        817.
                        818.
000976                  819. wstart:          * a0=Fehlertext
000976  6100 F730       820.  bsr print       * Fehler ausgeben
00097A  41FA 032D       821.  lea weiter(pc),a0
00097E  6100 F728       822.  bsr print
                        823.  repeat
000982  6100 F4B0       824.     bsr ci
000986  B03C 0077       825.  until.b d0 <eq> #'w' or.b d0 <eq> #'W'
        6706 B03C 0057
        66F0
000992  2E6E 0000       826.  movea.l oldstack(a6),a7 * neuer Stack, wieder bereinigt
000996  6000 002A       827.  bra start       * dort wieder aufsetzen danach.
                        828.
00099A                  829. kstart:
00099A  41F9 00008000   830.  lea $8000,a0    * dort Suche beginnen
0009A0  6100 F6D2       831.  bsr getram      * normalerweise ab 8000-9fff
                        832.                  * a0-> erste Nicht-Ram-Zelle
                        833.                  * a1-> erste RAM-Zelle
0009A4  6508            834.  bcs ramerror    * Fehler kein RAM da
0009A6  2009            835.  move.l a1,d0    * erste RAM-Zelle
0009A8  0880 0000       836.  bclr #0,d0      * muss EVEN sein
0009AC  D0BC 00001000   837.  add.l #$1000,d0 * ab $9000, wenn bei $8000 RAM
0009B2  2C40            838.  movea.l d0,a6   * a6 ist ab sofort RAM Pointer
0009B4  D0BC 00001C30   839.  add.l #finram,d0 * Ende des Rams
```

```
0009BA  B088            840.  cmp.l a0,d0     * ramende > ram
0009BC  64A0            841.  bcc ramerror    * sonst RAM-Platz zu klein.
                        842.                  * ok, Start des Hauptprogramms
0009BE  2D4F 0000       843.  move.l a7,oldstack(a6) * Stackpointer festhalten
0009C2                  844. start:           * Einsprung Hauptschleife
0009C2  303C 0021       845.  move #$21,d0    * Grosse Schrift
0009C6  6100 F69C       846.  bsr size
0009CA  6100 F678       847.  bsr clrscreen   * und Bildschirm loeschen
0009CE  41FA 0601       848.  lea headtxt(pc),a0
0009D2  6100 F6D4       849.  bsr print
0009D6                  850. stilp:
0009D6  6100 F65C       851.   bsr ci
0009DA  B03C 0031       852.   cmp.b #'1',d0
0009DE  65F6            853.   bcs stilp
0009E0  B03C 0037       854.   cmp.b #'6'+1,d0
0009E4  64F0            855.   bcc stilp
                        856. *
0009E6  422E 0008       857.  clr.b ecma(a6)
0009EA  422E 000F       858.  clr.b sidesel(a6)
0009EE  422E 0010       859.  clr.b sideflag(a6)
0009F2  B03C 0031       860.  if.b d0 <eq> #'1' then * Mini
        6600 001A
0009FA  422E 0007       861.     clr.b minimax(a6)
0009FE  6100 FB16       862.     bsr setspur
000A02  6100 FB62       863.     bsr setlw
000A06  6100 FB9C       864.     bsr setdichte
000A0A  6100 FBC4       865.     bsr setsek
000A0E  6000 00DA       866.  else if.b d0 <eq> #'2' then * Maxi
        B03C 0032 6600
        001C
000A1A  1D7C 0001       867.     move.b #1,minimax(a6)
        0007
000A20  6100 FAF4       868.     bsr setspur
000A24  6100 FB40       869.     bsr setlw
000A28  6100 FB7A       870.     bsr setdichte
000A2C  6100 FBA2       871.     bsr setsek
000A30  6000 00B8       872.  else if.b d0 <eq> #'3' then * 8 Zoll
        B03C 0033 6600
        002C
000A3C  1D7C 0001       873.     move.b #1,minimax(a6)
        0007
000A42  3D7C 0080       874.     move.w #128,anzbyte(a6)
        0004
000A48  422E 0009       875.     clr.b dichte(a6)
000A4C  1D7C 001B       876.     move.b #27,gaplen(a6)
        000A
000A52  422E 000C       877.     clr.b kopf(a6)
000A56  1D7C 001A       878.     move.b #26,sektoren(a6)
        000D
000A5C  1D7C 004D       879.     move.b #77,spuren(a6)
        000B
000A62  6000 0086       880.  else if.b d0 <eq> #'4' then * Ecma
        B03C 0034 6600
        0038
000A6E  422E 0007       881.     clr.b minimax(a6)
000A72  1D7C 0001       882.     move.b #1,ecma(a6)
        0008
000A78  1D7C 0001       883.     move.b #1,sideflag(a6) * Rueckseite mit Sidebit
        0010
```

```
000A7E  3D7C 0100       884.     move.w #256,anzbyte(a6)
        0004
000A84  1D7C 0001       885.     move.b #1,dichte(a6)
        0009
000A8A  422E 000C       886.     clr.b kopf(a6)
000A8E  1D7C 0010       887.     move.b #16,sektoren(a6)
        000D
000A94  1D7C 0036       888.     move.b #54,gaplen(a6)
        000A
000A9A  1D7C 0028       889.     move.b #40,spuren(a6)
        000B
000AA0  6000 0048       890.  else if.b d0 <eq> #'5' then * NDR 80 Spur
        B03C 0035 6600
        0034
000AAC  422E 0007       891.     clr.b minimax(a6)
000AB0  3D7C 0400       892.     move #1024,anzbyte(a6)
        0004
000AB6  1D7C 0001       893.     move.b #1,dichte(a6)
        0009
000ABC  1D7C 0036       894.     move.b #54,gaplen(a6)
        000A
000AC2  1D7C 0005       895.     move.b #5,sektoren(a6)
        000D
000AC8  1D7C 0001       896.     move.b #1,kopf(a6)
        000C
000ACE  1D7C 0001       897.     move.b #1,sideflag(a6)
        0010
000AD4  1D7C 0050       898.     move.b #80,spuren(a6)
        000B
000ADA  6000 000E       899.  else if.b d0 <eq> #'6' then * Ende
        B03C 0036 6600
        0006
000AE6  6000 0108       900.     bra exit
                        901.  endi
                        902. *
000AEA                  903. inner:                  * Start danach
000AEA  41FA 0973       904.  lea menue6(pc),a0
000AEE  6100 F588       905.  bsr print              * Laufwerk eingeben
000AF2  6100 F540       906.  bsr ci
000AF6  B03C 0031       907.  if.b d0 <eq> #'1' then
        6600 000A
000AFE  103C 0001       908.     move.b #1,d0
000B02  6000 0096       909.  else if.b d0 <eq> #'2' then
        B03C 0032 6600
        000A
000B0E  103C 0002       910.     move.b #2,d0
000B12  6000 0086       911.  else if.b d0 <eq> #'3' then
        B03C 0033 6600
        000A
000B1E  103C 0004       912.     move.b #4,d0
000B22  6000 0076       913.  else if.b d0 <eq> #'4' then
        B03C 0034 6600
        000A
000B2E  103C 0008       914.     move.b #8,d0
000B32  6000 0066       915.  else if.b d0 <eq> #'5' then
        B03C 0035 6600
        000A
000B3E  103C 0081       916.     move.b #$81,d0
000B42  6000 0056       917.  else if.b d0 <eq> #'6' then
        B03C 0036 6600
        000A
```

```
000B4E  103C 0082       918.     move.b #$82,d0
000B52  6000 0046       919.  else if.b d0 <eq> #'7' then
        B03C 0037 6600
        000A
000B5E  103C 0084       920.     move.b #$84,d0
000B62  6000 0036       921.  else if.b d0 <eq> #'8' then
        B03C 0038 6600
        000A
000B6E  103C 0088       922.     move.b #$88,d0
000B72  6000 0026       923.  else if.b d0 <eq> #'9' then
        B03C 0039 6600
        000A
000B7E  6000 FE42       924.     bra start
000B82  6000 0016       925.  else if.b d0 <eq> #'0' then
        B03C 0030 6600
        000A
000B8E  6000 0060       926.     bra exit
000B92  6000 0006       927.  else
000B96  6000 FF52       928.     bra inner
                        929.  endi
000B9A  1D40 000E       930.  move.b d0,laufwerk(a6)
000B9E  4A2E 0010       931.  tst.b sideflag(a6)
000BA2  670E            932.  if <ne> then.s
000BA4  102E 000E       933.     move.b laufwerk(a6),d0
000BA8  C03C 0080       934.     and.b #$80,d0
000BAC  6704            935.     if <ne> then.s        * wenn flag, dann
000BAE  1D40 000F       936.        move.b d0,sidesel(a6) * rueckseite mit sidebit
                        937.     endi
                        938.  endi
000BB2                  939. st2lpa:
000BB2  6100 FC18       940.   bsr werteaus          * Kenndaten ausgeben
000BB6  41FA 09B8       941.   lea formtxt(pc),a0
000BBA  6100 F4EC       942.   bsr print
000BBE  6100 F474       943.   bsr ci
000BC2  B03C 004A       944.   cmp.b #'J',d0
000BC6  6720            945.   beq.s st3lpm
000BC8  B03C 006A       946.   cmp.b #'j',d0
000BCC  671A            947.   beq.s st3lpm
000BCE  B03C 004E       948.   cmp.b #'N',d0
000BD2  6700 FF16       949.   beq inner
000BD6  B03C 006E       950.   cmp.b #'n',d0
000BDA  6700 FF0E       951.   beq inner
000BDE  B03C 0003       952.   cmp.b #3,d0
000BE2  6700 000C       953.   beq exit
000BE6  60CA            954.   bra st2lpm
                        955.
000BE8                  956. st3lpm:          * Formatierung beginnt
000BE8  6100 F5A8       957.  bsr format
000BEC  6000 FEFC       958.  bra inner       * und wieder zurueck
                        959.
                        960.
000BF0                  961. exit:
000BF0  41FA 03C3       962.  lea fimsg(pc),a0
000BF4  6100 F4B2       963.  bsr print
000BF8  4216            964.  clr.b (a6)      * ggf. Textanfang loeschen.
000BFA  4E75            965.  rts             * Ende
                        966.
                        967. *** Texte
                        968.
```

```
000BFC                          969. txterr:
000BFC  0D 0A 2A 2A             970.  dc.b $d,$a,'*** Fehler, zuwenig RAM ***',$d,$a
        2A 20 46 65 68
        6C 65 72 2C 20
        7A 75 77 65 6E
        69 67 20 52 41
        4D 20 2A 2A 2A
        0D 0A
000C18  0D 0A 44 61             971.  dc.b $d,$a,'Das Programm benoetigt mindestens',$d,$a
        73 20 50 72 6F
        67 72 61 6D 6D
        20 62 65 6E 6F
        65 74 69 67 74
        20 6D 69 6E 64
        65 73 74 65 6E
        73 0D 0A
000C40  0D 0A 38 4B             972.  dc.b $d,$a,'8K eigenen RAM, beginnend beim ',$d,$a
        20 65 69 67 65
        6E 65 6E 20 52
        41 4D 2C 20 62
        65 67 69 6E 6E
        65 6E 64 20 62
        65 69 6D 20 0D
        0A
000C63  0D 0A 54 65             973.  dc.b $d,$a,'Textspeicher, der ueberschrieben',$d,$a
        78 74 73 70 65
        69 63 68 65 72
        2C 20 64 65 72
        20 75 65 62 65
        72 73 63 68 72
        69 65 62 65 6E
        0D 0A
000C87  0D 0A 77 69             974.  dc.b $d,$a,'wird.',$d,$a
        72 64 2E 0D 0A
000C90  42 69 74 74             975.  dc.b 'Bitte  RESET druecken.',$d,$a,0
        65 20 20 52 45
        53 45 54 20 64
        72 75 65 63 6B
        65 6E 2E 0D 0A
        00
                                976.
000CA9                          977. weiter:
000CA9  0D 0A 77 65             978.  dc.b $d,$a,'weiter = w ',0
        69 74 65 72 20
        3D 20 77 20 00
                                979.
000CB7                          980. rmsg:
000CB7  0D 0A                   981.  dc.b $d,$a
000CB9  46 65 68 6C             982.  dc.b 'Fehler: Laufwerk meldet nicht READY.',$d,$a
        65 72 3A 20 4C
        61 75 66 77 65
        72 6B 20 6D 65
        6C 64 65 74 20
        6E 69 63 68 74
        20 52 45 41 44
        59 2E 0D 0A
000CDF  42 69 74 74             983.  dc.b 'Bitte pruefen:',$d,$a
        65 20 70 72 75
        65 66 65 6E 3A
        0D 0A
```

```
000CEF  20 31 2E 20      984.  dc.b ' 1. Diskette falsch eingelegt.',$d,$a
        44 69 73 6B 65
        74 74 65 20 66
        61 6C 73 63 68
        20 65 69 6E 67
        65 6C 65 67 74
        2E 0D 0A
000D0F  20 32 2E 20      985.  dc.b ' 2. Laufwerktuere offen.',$d,$a
        4C 61 75 66 77
        65 72 6B 74 75
        65 72 65 20 6F
        66 66 65 6E 2E
        0D 0A
000D29  20 33 2E 20      986.  dc.b ' 3. Falsche Laufwerksnummer.',$d,$a
        46 61 6C 73 63
        68 65 20 4C 61
        75 66 77 65 72
        6B 73 6E 75 6D
        6D 65 72 2E 0D
        0A
000D47  20 34 2E 20      987.  dc.b ' 4. Motor dreht sich nicht.',$d,$a
        4D 6F 74 6F 72
        20 64 72 65 68
        74 20 73 69 63
        68 20 6E 69 63
        68 74 2E 0D 0A
000D64  20 35 2E 20      988.  dc.b ' 5. READY-Leitung nicht ok.',$d,$a,0
        52 45 41 44 59
        2D 4C 65 69 74
        75 6E 67 20 6E
        69 63 68 74 20
        6F 6B 2E 0D 0A
        00
                         989.
000D82                   990. fmsg:
000D82  1A 0D 0A         991.  dc.b $1a,$d,$a
000D85  44 69 73 6B      992.  dc.b 'Diskette wird formatiert. Bitte warten.',$d,$a
        65 74 74 65 20
        77 69 72 64 20
        66 6F 72 6D 61
        74 69 65 72 74
        2E 20 42 69 74
        74 65 20 77 61
        72 74 65 6E 2E
        0D 0A
000DAE  0D 0A 00         993.  dc.b $d,$a,0
                         994.
000DB1                   995. f6msg:
000DB1  0D 0A            996.  dc.b $d,$a
000DB3  46 65 68 6C      997.  dc.b 'Fehler: Schreibschutz gesetzt',$d,$a
        65 72 3A 20 53
        63 68 72 65 69
        62 73 63 68 75
        74 7A 20 67 65
        73 65 74 7A 74
        0D 0A
000DD2  44 65 72 20      998.  dc.b 'Der Fehler entsteht auch, wenn',$d,$a
        46 65 68 6C 65
        72 20 65 6E 74
```

```
        73 74 65 68 74
        20 61 75 63 68
        2C 20 77 65 6E
        6E 0D 0A
000DF2  64 69 65 20      999.  dc.b 'die Diskette falsch herum ',$d,$a
        44 69 73 6B 65
        74 74 65 20 66
        61 6C 73 63 68
        20 68 65 72 75
        6D 20 0D 0A
000E0E  65 69 6E 67     1000.  dc.b 'eingelegt wurde, oder die',$d,$a
        65 6C 65 67 74
        20 77 75 72 64
        65 2C 20 6F 64
        65 72 20 64 69
        65 0D 0A
000E29  4C 61 75 66     1001.  dc.b 'Laufwerkstuere offen ist',$d,$a
        77 65 72 6B 73
        74 75 65 72 65
        20 6F 66 66 65
        6E 20 69 73 74
        0D 0A
000E43  75 6E 64 20     1002.  dc.b 'und das Laufwerk dennoch',$d,$a
        64 61 73 20 4C
        61 75 66 77 65
        72 6B 20 64 65
        6E 6E 6F 63 68
        0D 0A
000E5D  52 45 41 44     1003.  dc.b 'READY meldet.'
        59 20 6D 65 6C
        64 65 74 2E
000E6A  0D 0A 00        1004.  dc.b $d,$a,0
                        1005.
000E6D                  1006. f5msg:
000E6D  0D 0A           1007.  dc.b $d,$a
000E6F  46 65 68 6C     1008.  dc.b 'Fehler: Record Typ ist falsch.',$d,$a,0
        65 72 3A 20 52
        65 63 6F 72 64
        20 54 79 70 20
        69 73 74 20 66
        61 6C 73 63 68
        2E 0D 0A 00
                        1009.
000E90                  1010. f4msg:
000E90  0D 0A           1011.  dc.b $d,$a
000E92  46 65 68 6C     1012.  dc.b 'Fehler: Record nicht gefunden.',$d,$a,0
        65 72 3A 20 52
        65 63 6F 72 64
        20 6E 69 63 68
        74 20 67 65 66
        75 6E 64 65 6E
        2E 0D 0A 00
                        1013.
000EB3                  1014. f3msg:
000EB3  0D 0A           1015.  dc.b $d,$a
000EB5  46 65 68 6C     1016.  dc.b 'Fehler: CRC-Fehler aufgetreten.',$d,$a,0
        65 72 3A 20 43
        52 43 2D 46 65
        68 6C 65 72 20
```

```
        61 75 66 67 65
        74 72 65 74 65
        6E 2E 0D 0A 00
                              1017.
000ED7                        1018. f2msg:
000ED7  0D 0A                 1019.  dc.b $d,$a
000ED9  43 50 55 20           1020.  dc.b 'CPU zu langsam fuer gewaehltes Format,',$d,$a
        7A 75 20 6C 61
        6E 67 73 61 6D
        20 66 75 65 72
        20 67 65 77 61
        65 68 6C 74 65
        73 20 46 6F 72
        6D 61 74 2C 0D
        0A
000F01  44 61 74 65           1021.  dc.b 'Datenverlust. ',$d,$a
        6E 76 65 72 6C
        75 73 74 2E 20
        0D 0A
000F11  48 6F 65 68           1022.  dc.b 'Hoehere Taktfrequenz noetig.',$d,$a
        65 72 65 20 54
        61 6B 74 66 72
        65 71 75 65 6E
        7A 20 6E 6F 65
        74 69 67 2E 0D
        0A
000F2F  4F 64 65 72           1023.  dc.b 'Oder Anzahl der Wait Zyklen zu gross.',$d,$a,0
        20 41 6E 7A 61
        68 6C 20 64 65
        72 20 57 61 69
        74 20 5A 79 6B
        6C 65 6E 20 7A
        75 20 67 72 6F
        73 73 2E 0D 0A
        00
                              1024.
000F57                        1025. f0msg:
000F57  0D 0A                 1026.  dc.b $d,$a
000F59  46 65 68 6C           1027.  dc.b 'Fehler aufgetreten',$d,$a,0
        65 72 20 61 75
        66 67 65 74 72
        65 74 65 6E 0D
        0A 00
                              1028.
000F6E                        1029. tkmsg:
000F6E  0D 0A 20 53           1030.  dc.b $d,$a,' Spur = ',0
        70 75 72 20 3D
        20 00
000F79                        1031. skmsg:
000F79  0D 0A 20 53           1032.  dc.b $d,$a,' Sektor = ',0
        65 6B 74 6F 72
        20 3D 20 00
000F86                        1033. tcrlf:
000F86  0D 0A 00              1034.  dc.b $d,$a,0
                              1035.
000F89                        1036. vmsg:
000F89  0D 0A                 1037.  dc.b $d,$a
000F8B  44 69 73 6B           1038.  dc.b 'Diskette wird geprueft. Bitte warten.',$d,$a
        65 74 74 65 20
```

```
        77 69 72 64 20
        67 65 70 72 75
        65 66 74 2E 20
        42 69 74 74 65
        20 77 61 72 74
        65 6E 2E 0D 0A
000FB2  0D 0A 00        1039.  dc.b $d,$a,0
                        1040.
000FB5                  1041. finsg:
000FB5  0D 0A 46 6F     1042.  dc.b $d,$a,'Formatprogramm beendet.',$d,$a,0
        72 6D 61 74 70
        72 6F 67 72 61
        6D 6D 20 62 65
        65 6E 64 65 74
        2E 0D 0A 00
                        1043.
                        1044.
                        1045.
000FD1                  1046. headtxt:
000FD1  1A 0D 0A        1047.  dc.b $1a,$d,$a
000FD4  55 6E 69 76     1048.  dc.b 'Universal Formatierer V 1.2 ',$d,$a
        65 72 73 61 6C
        20 46 6F 72 6D
        61 74 69 65 72
        65 72 20 56 20
        31 2E 32 20 0D
        0A
000FF2  28 43 29 20     1049.  dc.b '(C) 1985 Rolf-Dieter Klein  ',$d,$a
        31 39 38 35 20
        52 6F 6C 66 2D
        44 69 65 74 65
        72 20 4B 6C 65
        69 6E 20 20 0D
        0A
001010  0D 0A           1050.  dc.b $d,$a
001012  2D 2D 2D 20     1051.  dc.b '         Diskette          ',$d,$a
        20 20 20 20 20
        44 69 73 6B 65
        74 74 65 20 20
        20 20 20 2D 2D
        2D 20 20 0D 0A
00102F  2D 2D 2D 20     1052.  dc.b '         einlegen          ',$d,$a
        20 20 20 20 20
        65 69 6E 6C 65
        67 65 6E 20 20
        20 20 20 2D 2D
        2D 20 20 0D 0A
00104C  0D 0A           1053.  dc.b $d,$a
00104E  4D 69 6E 69     1054.  dc.b 'Minilaufwerk (5 1/4") = 1',$d,$a
        6C 61 75 66 77
        65 72 6B 20 28
        35 20 31 2F 34
        22 29 20 3D 20
        31 0D 0A
001069  4D 61 78 69     1055.  dc.b 'Maxilaufwerk (8"    ) = 2',$d,$a
        6C 61 75 66 77
        65 72 6B 20 28
        38 22 20 20 20
        20 29 20 3D 20
        32 0D 0A
```

```
001084 53 74 64 20     1056.  dc.b 'Std 8"  SD SS 77 Spur = 3',$d,$a
       38 22 20 20 53
       44 20 53 53 20
       37 37 20 53 70
       75 72 20 3D 20
       33 0D 0A
00109F 45 43 4D 41     1057.  dc.b 'ECMA 70 DD SS 40 Spur = 4',$d,$a
       20 37 30 20 44
       44 20 53 53 20
       34 30 20 53 70
       75 72 20 3D 20
       34 0D 0A
0010BA 4E 44 52 38     1058.  dc.b 'NDR80   DD DS 80 Spur = 5',$d,$a
       30 20 20 20 44
       44 20 44 53 20
       38 30 20 53 70
       75 72 20 3D 20
       35 0D 0A
0010D5 45 4E 44 45     1059.  dc.b 'ENDE                  = 6',$d,$a
       20 20 20 20 20
       20 20 20 20 20
       20 20 20 20 20
       20 20 20 3D 20
       36 0D 0A
0010F0 00              1060.  dc.b 0
                       1061.
0010F1                 1062. menue2:
0010F1 1A 0D 0A        1063.   dc.b $1a,$d,$a
0010F4 65 69 6E 66     1064.  dc.b 'einfache Schreibdichte = 1',$d,$a
       61 63 68 65 20
       53 63 68 72 65
       69 62 64 69 63
       68 74 65 20 3D
       20 31 0D 0A
001110 64 6F 70 70     1065.  dc.b 'doppelte Schreibdichte = 2',$d,$a,$d,$a,0
       65 6C 74 65 20
       53 63 68 72 65
       69 62 64 69 63
       68 74 65 20 3D
       20 32 0D 0A 0D
       0A 00
                       1066.
00112F                 1067. menue3:
00112F 1A 0D 0A        1068.  dc.b $1a,$d,$a.
001132 33 35 20 53     1069.  dc.b '35 Spuren = 1',$d,$a
       70 75 72 65 6E
       20 3D 20 31 0D
       0A
001141 34 30 20 53     1070.  dc.b '40 Spuren = 2',$d,$a
       70 75 72 65 6E
       20 3D 20 32 0D
       0A
001150 37 30 20 53     1071.  dc.b '70 Spuren = 3',$d,$a
       70 75 72 65 6E
       20 3D 20 33 0D
       0A
00115F 37 37 20 53     1072.  dc.b '77 Spuren = 4',$d,$a
       70 75 72 65 6E
       20 3D 20 34 0D
       0A
```

```
00116E  38 30 20 53     1073.  dc.b '80 Spuren = 5',$d,$a
        70 75 72 65 6E
        20 3D 20 35 0D
        0A
00117D  0D 0A 00        1074.  dc.b $d,$a,0
                        1075.
001180                  1076. menue4:
001180  1A 0D 0A        1077.  dc.b $1a,$d,$a
001183  45 69 6E 73     1078.  dc.b 'Einseitiges Laufwerk     = 1',$d,$a
        65 69 74 69 67
        65 73 20 4C 61
        75 66 77 65 72
        6B 20 20 20 20
        20 3D 20 31 0D
        0A
0011A1  44 6F 70 70     1079.  dc.b 'Doppeltseitiges Laufwerk = 2',$d,$a
        65 6C 74 73 65
        69 74 69 67 65
        73 20 4C 61 75
        66 77 65 72 6B
        20 3D 20 32 0D
        0A
0011BF  44 6F 70 70     1080.  dc.b 'Doppeltseitig und SSO    = 3',$d,$a
        65 6C 74 73 65
        69 74 69 67 20
        75 6E 64 20 53
        53 4F 20 20 20
        20 3D 20 33 0D
        0A
0011DD  0D 0A 00        1081.  dc.b $d,$a,0
                        1082.
0011E0                  1083. menue5a:
0011E0  1A 0D 0A        1084.  dc.b $1a,$d,$a
0011E3  31 32 38 20     1085.  dc.b '128 Bytes pro Sektor (16 pro Track) = 1',$d,$a
        42 79 74 65 73
        20 70 72 6F 20
        53 65 6B 74 6F
        72 20 28 31 36
        20 70 72 6F 20
        54 72 61 63 6B
        29 20 3D 20 31
        0D 0A
00120C  31 32 38 20     1086.  dc.b '128 Bytes pro Sektor (18 pro Track) = 2',$d,$a
        42 79 74 65 73
        20 70 72 6F 20
        53 65 6B 74 6F
        72 20 28 31 38
        20 70 72 6F 20
        54 72 61 63 6B
        29 20 3D 20 32
        0D 0A
001235  32 35 36 20     1087.  dc.b '256 Bytes pro Sektor (10 pro Track) = 3',$d,$a
        42 79 74 65 73
        20 70 72 6F 20
        53 65 6B 74 6F
        72 20 28 31 30
        20 70 72 6F 20
        54 72 61 63 6B
        29 20 3D 20 33
        0D 0A
```

```
00125E  0D 0A 00        1088.  dc.b $d,$a,0
                        1089.
001261                  1090. menue5b:
001261  1A 0D 0A        1091.  dc.b $1a,$d,$a
001264  32 35 36 20     1092.  dc.b '256 Bytes pro Sektor (16 pro Track) = 1',$d,$a
        42 79 74 65 73
        20 70 72 6F 20
        53 65 6B 74 6F
        72 20 28 31 36
        20 70 72 6F 20
        54 72 61 63 6B
        29 20 3D 20 31
        0D 0A
00128D  35 31 32 20     1093.  dc.b '512 Bytes pro Sektor ( 9 pro Track) = 2',$d,$a
        42 79 74 65 73
        20 70 72 6F 20
        53 65 6B 74 6F
        72 20 28 20 39
        20 70 72 6F 20
        54 72 61 63 6B
        29 20 3D 20 32
        0D 0A
0012B6  35 31 32 20     1094.  dc.b '512 Bytes pro Sektor (10 pro Track) = 3',$d,$a
        42 79 74 65 73
        20 70 72 6F 20
        53 65 6B 74 6F
        72 20 28 31 30
        20 70 72 6F 20
        54 72 61 63 6B
        29 20 3D 20 33
        0D 0A
0012DF  31 30 32 34     1095.  dc.b '1024 Bytes pro Sektor( 5 pro Track) = 4',$d,$a
        20 42 79 74 65
        73 20 70 72 6F
        20 53 65 6B 74
        6F 72 28 20 35
        20 70 72 6F 20
        54 72 61 63 6B
        29 20 3D 20 34
        0D 0A
001308  0D 0A 00        1096.  dc.b $d,$a,0
                        1097.
001308                  1098. menue5c:
001308  1A 0D 0A        1099.  dc.b $1a,$d,$a
00130E  31 32 38 20     1100.  dc.b '128 Bytes pro Sektor (26 pro Track) = 1',$d,$a
        42 79 74 65 73
        20 70 72 6F 20
        53 65 6B 74 6F
        72 20 28 32 36
        20 70 72 6F 20
        54 72 61 63 6B
        29 20 3D 20 31
        0D 0A
001337  32 35 36 20     1101.  dc.b '256 Bytes pro Sektor (15 pro Track) = 2',$d,$a
        42 79 74 65 73
        20 70 72 6F 20
        53 65 6B 74 6F
        72 20 28 31 35
        20 70 72 6F 20
```

```
        54 72 61 63 6B
        29 20 3D 20 32
        0D 0A
001360  0D 0A 00        1102.  dc.b $d,$a,0
                        1103.
001363                  1104. menue5d:
001363  1A 0D 0A        1105.  dc.b $1a,$d,$a
001366  32 35 36 20     1106.  dc.b '256 Bytes pro Sektor (26 pro Track) = 1',$d,$a
        42 79 74 65 73
        20 70 72 6F 20
        53 65 6B 74 6F
        72 20 28 32 36
        20 70 72 6F 20
        54 72 61 63 6B
        29 20 3D 20 31
        0D 0A
00138F  35 31 32 20     1107.  dc.b '512 Bytes pro Sektor (14 pro Track) = 2',$d,$a
        42 79 74 65 73
        20 70 72 6F 20
        53 65 6B 74 6F
        72 20 28 31 34
        20 70 72 6F 20
        54 72 61 63 6B
        29 20 3D 20 32
        0D 0A
0013B8  35 31 32 20     1108.  dc.b '512 Bytes pro Sektor (15 pro Track) = 3',$d,$a
        42 79 74 65 73
        20 70 72 6F 20
        53 65 6B 74 6F
        72 20 28 31 35
        20 70 72 6F 20
        54 72 61 63 6B
        29 20 3D 20 33
        0D 0A
0013E1  35 31 32 20     1109.  dc.b '512 Bytes pro Sektor (16 pro Track) = 4',$d,$a
        42 79 74 65 73
        20 70 72 6F 20
        53 65 6B 74 6F
        72 20 28 31 36
        20 70 72 6F 20
        54 72 61 63 6B
        29 20 3D 20 34
        0D 0A
00140A  31 30 32 34     1110.  dc.b '1024 Bytes pro Sektor( 8 pro Track) = 5',$d,$a
        20 42 79 74 65
        73 20 70 72 6F
        20 53 65 6B 74
        6F 72 28 20 38
        20 70 72 6F 20
        54 72 61 63 6B
        29 20 3D 20 35
        0D 0A
001433  31 30 32 34     1111.  dc.b '1024 Bytes pro Sektor( 9 pro Track) = 6',$d,$a
        20 42 79 74 65
        73 20 70 72 6F
        20 53 65 6B 74
        6F 72 28 20 39
        20 70 72 6F 20
        54 72 61 63 6B
```

```
        29 20 3D 20 36
        0D 0A
00145C  0D 0A 00        1112.  dc.b $d,$a,0
                        1113.
00145F                  1114. aenue6:
00145F  1A 0D 0A        1115.  dc.b $1a,$d,$a
001462  4C 61 75 66     1116.  dc.b 'Laufwerk            A = 1',$d,$a
        77 65 72 6B 20
        20 20 20 20 20
        20 20 20 20 20
        20 41 20 3D 20
        31 0D 0A
00147D  4C 61 75 66     1117.  dc.b 'Laufwerk            B = 2',$d,$a
        77 65 72 6B 20
        20 20 20 20 20
        20 20 20 20 20
        20 42 20 3D 20
        32 0D 0A
001498  4C 61 75 66     1118.  dc.b 'Laufwerk            C = 3',$d,$a
        77 65 72 6B 20
        20 20 20 20 20
        20 20 20 20 20
        20 43 20 3D 20
        33 0D 0A
0014B3  4C 61 75 66     1119.  dc.b 'Laufwerk            D = 4',$d,$a
        77 65 72 6B 20
        20 20 20 20 20
        20 20 20 20 20
        20 44 20 3D 20
        34 0D 0A
0014CE  4C 61 75 66     1120.  dc.b 'Laufwerk Rueckseite A = 5',$d,$a
        77 65 72 6B 20
        52 75 65 63 6B
        73 65 69 74 65
        20 41 20 3D 20
        35 0D 0A
0014E9  4C 61 75 66     1121.  dc.b 'Laufwerk Rueckseite B = 6',$d,$a
        77 65 72 6B 20
        52 75 65 63 6B
        73 65 69 74 65
        20 42 20 3D 20
        36 0D 0A
001504  4C 61 75 66     1122.  dc.b 'Laufwerk Rueckseite C = 7',$d,$a
        77 65 72 6B 20
        52 75 65 63 6B
        73 65 69 74 65
        20 43 20 3D 20
        37 0D 0A
00151F  4C 61 75 66     1123.  dc.b 'Laufwerk Rueckseite D = 8',$d,$a
        77 65 72 6B 20
        52 75 65 63 6B
        73 65 69 74 65
        20 44 20 3D 20
        38 0D 0A
00153A  4E 45 55 53     1124.  dc.b 'NEUSTART              = 9',$d,$a
        54 41 52 54 20
        20 20 20 20 20
        20 20 20 20 20
```

```
        20 20 20 3D 20
        39 0D 0A
001555  45 4E 44 45    1125.  dc.b 'ENDE                    = 0',$d,$a
        20 20 20 20 20
        20 20 20 20 20
        20 20 20 20 20
        20 20 20 3D 20
        30 0D 0A
001570  0D 0A 00        1126.  dc.b $d,$a,0
                        1127.
001573                  1128. formtxt:
001573  0D 0A           1129.  dc.b $d,$a
001575  41 43 48 54     1130.  dc.b 'ACHTUNG. Diskette wird formatiert.',$d,$a
        55 4E 47 2E 20
        44 69 73 6B 65
        74 74 65 20 77
        69 72 64 20 66
        6F 72 6D 61 74
        69 65 72 74 2E
        0D 0A
001599  53 74 61 72     1131.  dc.b 'Start = J',$d,$a,0
        74 20 3D 20 4A
        0D 0A 00
                        1132.
0015A5                  1133. tkap:
0015A5  53 70 65 69     1134.  dc.b 'Speicherkapazitaet :  ',0
        63 68 65 72 6B
        61 70 61 7A 69
        74 61 65 74 20
        3A 20 20 00
0015BC                  1135. tkap1:
0015BC  20 4B 42 79     1136.  dc.b ' KBytes',0
        74 65 73 00
                        1137.
0015C4                  1138. tmini:
0015C4  4D 69 6E 69     1139.  dc.b 'Mini-Laufwerk  ',0
        2D 4C 61 75 66
        77 65 72 6B 20
        20 00
0015D4                  1140. tmaxi:
0015D4  4D 61 78 69     1141.  dc.b 'Maxi Laufwerk  ',0
        20 4C 61 75 66
        77 65 72 6B 20
        20 00
0015E4                  1142. tfm:
0015E4  46 4D 20 20     1143.  dc.b 'FM  ',0
        00
0015E9                  1144. tmfm:
0015E9  4D 46 4D 20     1145.  dc.b 'MFM  ',0
        20 00
0015EF                  1146. tecma:
0015EF  45 43 4D 41     1147.  dc.b 'ECMA70 ',0
        37 30 20 00
0015F7                  1148. tspuren:
0015F7  41 6E 7A 61     1149.  dc.b 'Anzahl Spuren    = ',0
        68 6C 20 53 70
        75 72 65 6E 20
        20 20 20 3D 20
        00
```

```
001608                  1150. tsektoren:
001608  41 6E 7A 61     1151.  dc.b 'Anzahl Sektoren  = ',0
        68 6C 20 53 65
        6B 74 6F 72 65
        6E 20 20 3D 20
        00
00161F                  1152. tbytes:
00161F  42 79 74 65     1153.  dc.b 'Bytes/pro Sektor = ',0
        73 2F 70 72 6F
        20 53 65 6B 74
        6F 72 20 3D 20
        00
001633                  1154. tlaufwerk:
001633  4C 61 75 66     1155.  dc.b 'Laufwerk ',0
        77 65 72 6B 20
        00
00163D                  1156. teinseitig:
00163D  56 6F 72 64     1157.  dc.b 'Vorderseite wird formatiert',0
        65 72 73 65 69
        74 65 20 77 69
        72 64 20 66 6F
        72 6D 61 74 69
        65 72 74 00
001659                  1158. trueckseite:
001659  52 75 65 63     1159.  dc.b 'Rueckseite wird formatiert',0
        6B 73 65 69 74
        65 20 77 69 72
        64 20 66 6F 72
        6D 61 74 69 65
        72 74 00
001674                  1160. truecksso:
001674  52 75 65 63     1161.  dc.b 'Rueckseite wird mit SSO formatiert',0
        6B 73 65 69 74
        65 20 77 69 72
        64 20 6D 69 74
        20 53 53 4F 20
        66 6F 72 6D 61
        74 69 65 72 74
        00
001697                  1162. tzweiseitig:
001697  62 65 69 64     1163.  dc.b 'beide Seiten werden formatiert',0
        65 20 53 65 69
        74 65 6E 20 77
        65 72 64 65 6E
        20 66 6F 72 6D
        61 74 69 65 72
        74 00
0016B6                  1164. tzweisso:
0016B6  62 65 69 64     1165.  dc.b 'beide Seiten werden formatiert',$d,$a
        65 20 53 65 69
        74 65 6E 20 77
        65 72 64 65 6E
        20 66 6F 72 6D
        61 74 69 65 72
        74 0D 0A
0016D6  53 65 69 74     1166.  dc.b 'Seite 1 mit SSO, Seite 0 normal',0
        65 20 31 20 6D
        69 74 20 53 53
        4F 2C 20 53 65
```

```
        69 74 65 20 30
        20 6E 6F 72 6D
        61 6C 00
                          1167.
0016F6                    1168. ende:
                          1169.
                          1170.
0016F6                    1171. end
                          1172.
                          1173.
    0 Errors detected
```

B Zusammenstellung der Befehle des Grundprogramms

Inhaltsverzeichnis

Befehl	Index	Bedeutung
ADJ360	83	360 Grad Bereich
ASSEMBLE	65	Assembler starten
ASSERR	120	Fehlerzahl
AUFK	94	Schildkrötenrichtung setzen
AUFXY	92	Schildkröte setzen
AUTOFLIP	60	Bildseitenumschaltung
CHAR	63	Zeichen ausgeben
CI	12	Zeichen einlesen
CI2	32	Zeichen aus Speicher lesen
CIINIT2	31	Zeiger auf Textstart setzen
CLPG	17	Eine Bildschirmseite löschen
CLR	16	Alle Bildschirmseiten löschen
CLRSCREEN	20	Bildschirm löschen und Cursor setzen
CMD	26	Einen Befehl an GDP ausgeben
CMDPRINT	40	Befehlsausgabe mehrere Befehle an den GDP
CO	21	Zeichenausgabe auf den Bildschirm
CO2	33	Zeichen- und Steuerzeichenausgabe
COS	24	Cosinus*256 bilden
CRLF	99	Zeilenvorschub ausgeben
CRT	49	CO2 auf Bildschirm
CSTS	13	Zeichen da ?
CUROFF	82	Cursor ausblenden bei CO,CO2
CURON	81	Cursor erscheint bei CO,CO2
CURSAUS	62	Cursor ausblenden
CURSEIN	61	Cursor einblenden

DELAY	35	Warteschleife (Synchron zu Quarz)
DIVS32	73	32–Bit–Division
DRAWTO	9	Linie zur neuen X,Y–Position zeichnen
DREHE	2	Schildkröte dreht sich
EDIT	56	Editor aufrufen
ERAPEN	38	Schreiber in GDP auf Löschen
FIGUR	57	Figur–Befehl
FIRSTTIME	36	Schildkrötenneustart
FLINIT	74	Floppy–Variable definieren
FLOPPY	75	Floppy–Befehl ausführen
GETAD8	109	Wert vom 8 Bit AD–Umsetzer laden
GETAD10	110	Wert vom 10 Bit AD–Umsetzer laden
GETBASIS	89	Adresse des Grundprogramms laden
GETCOLOR	80	Farbcode laden
GETCURXY	101	Cursorposition laden
GETERR	54	Fehlercode lesen
GETFLOP	76	Floppy–Format feststellen
GETK	95	Schildkrötenrichtung laden
GETLINE	100	Zeilenadresse, Bildwidh. laden
GETNEXT	87	Offset für Symboltabelle
GETORG	68	Default–ORG–adresse laden
GETRAM	59	Speicherbereiche testen
GETSN	98	Seriennummer laden
GETSTX	66	Textstart–Adresse laden
GETSYM	86	Adresse, Symboltabelle laden
GETUHR	115	Uhrzeit und Datum einlesen
GETVAR	90	Adresse des Arbeitsspeichers laden
GETVERS	97	Version des Grundprogramms laden
GETXOR	78	Spezial–Mode laden
GETXY	103	Aktuelle GDP–Position laden
GRAPOFF	39	Schildkrötengraphik aus
HEBE	3	Schreibspur ausschalten
HIDE	47	Schildkröte ausblenden
KORXY	93	Schildkötenkoordinaten laden
LO	22	Zeichenausgabe auf den Drucker
LST	50	CO2 auf Drucker
LSTS	117	Drucker bereit ?
MOVETO	8	X,Y–Position im GDP setzen
MULS32	72	32–Bit–Multiplikation
NEWPAGE	27	Bildschirm Schreib– und Leseseite setzen
NIL	52	CO2 ignorieren
PO	15	Zeichen auf CAS–Baugruppe ausgeben
PUTNEXT	88	Offset für Symboltabelle setzen
PUTORG	69	Default–ORG–Adresse setzen
PUTSTX	67	Textstart neu festsetzen
PRINT8B	45	Binäre Ausgabe
PRINT4D	46	Dezimale Ausgabe

PRINT8D	70	Dezimale Ausgabe
PRINT2X	41	Sedezimale Ausgabe
PRINT4X	42	Sedezimale Ausgabe
PRINT6X	43	Sedezimale Ausgabe
PRINT8X	44	Sedezimale Ausgabe
PRINTV8D	71	Dezimale Ausgabe
PROGZGE	64	Zeichen frei wählbar, ausgeben
PRTSYM	84	Symboltabelle ausgeben
READ	11	Text einlesen
RELAN	118	Relais auf CAS2 einschalten
RELAUS	119	Relais auf CAS2 ausschalten
RI	14	Zeichen von CAS–Baugruppe einlesen
RND	96	Zufallszahl bestimmen
SCHR16TEL	19	Schrildköte schreitet 1/16
SCHREITE	1	Schildkröte schreitet
SETFLIP	34	Umschaltrate für AUTOFLIP setzen
SIN	23	Sinus*256 bilden
SIZE	25	Textgröße setzen
SENKE	4	Schreibspur wird sichtbar
SET	7	Schildkröte setzen, absolut
SETA5	91	Ram–Zeiger A5 gültig belegen
SETCOLOR	79	Farbcode setzen
SETCURXY	102	Cursorposition setzen
SETDA	111	DA–Umsetzer setzen
SETERR	53	Fehlercode setzen
SETFIG	58	Figur einfrieren
SETPASS	55	Laufnummer setzen
SETPEN	37	Schreiber in GDP auf Setzen
SETUHR	116	Uhrzeit und Datum neu stellen
SETXOR	77	Spezial–Mode setzen
SHOW	48	Schildkröte einblenden
SI	104	Zeichen von serieller Schnittstelle
SIINIT	108	Baudrate etc. setzen
SIN	23	Sinus*256 bilden
SISTS	106	Zeichen da?
SO	105	Zeichen an serielle Schnittstelle
SOSTS	107	Zeichen ausgegeben ?
SOUND	114	Sound–Generator setzen
SPEAK	112	Sprachausgabe mit 5 Parametern
SPEAK1	113	Sprachausgabe nur Phonemcode
SYMCLR	85	Symboltabelle löschen
SYNC	28	Vertikal–Synchronpuls testen
USR	51	CO2 auf Benutzerschnittstelle
WAIT	18	Warten bis GDP für Befehle bereit
WERT	29	Auswertung eines arithmetischen Ausdrucks
WRITE	10	Text auf dem Bildschirm ausgeben
ZUWEIS	30	Variable mit Wert eines Ausdrucks belegen

Befehlsübersicht

In der nachfolgenden Befehlsübersicht sind alle Befehle des Grundprogramms Version 4.2 aufgelistet. Dabei bedeutet (S): Schildkrötenbefehl.

Befehl	Index		Register
SCHREITE	1	(S)	D0
DREHE	2	(S)	D0
HEBE	3	(S)	
SENKE	4	(S)	
SET	7	(S)	D1 D2 D3
MOVETO	8		D1 D2
DRAWTO	9		D1 D2
WRITE	10		D0.B D1 D2 A0
READ	11		D0.B D1 D2 D3 A0 -> D4 D5
CI	12		->D0.L (-- .L ab Rev 4.2)
CSTS	13		->D0.L (-- sonst nur .B gueltig)
RI	14		->D0.L (--)
PO	15		D0.B
CLR	16		
CLPG	17		
WAIT	18		
SCHR16TEL	19	(S)	D0
CLRSCREEN	20		
CO	21		D0.B
LO	22		D0.B
SIN	23		D0 -> D0
COS	24		D0 -> D0
SIZE	25		D0.B
CMD	26		D0.B
NEWPAGE	27		D0 D1
SYNC	28		-> D0.B
WERT	29		A0 -> D0.L D1
ZUWEIS	30		A0 -> D0.L
CIINIT2	31		
CI2	32		-> D0.B
CO2	33		D0.B
SETFLIP	34		D0 D1
DELAY	35		D0.L
FIRSTTIME	36		
SETPEN	37		
ERAPEN	38		
GRAPOFF	39	(S)	
CMDPRINT	40		D0.B D1 D2 A0
PRINT2X	41		D0.B A0
PRINT4X	42		D0 A0
PRINT6X	43		D0.L A0
PRINT8X	44		D0.L A0
PRINT8B	45		D0.B A0

```
PRINT4D          46      D0 A0
HIDE             47  (S)
SHOW             48  (S)
CRT              49
LST              50
USR              51
NIL              52
SETERR           53      D0
GETERR           54      -> D0
SETPASS          55      D0
EDIT             56
FIGUR            57      D0 A0
SETFIG           58
GETRAM           59      -> A0 A1
AUTOFLIP         60
CURSEIN          61
CURSAUS          62
CHAR             63      D0.B
PROGZGR          64      A0
ASSEMBLE         65
GETSTX           66      -> D0.L
PUTSTX           67      D0.L
GETORG           68      -> D0.L
PUTORG           69      D0.L
PRINT8D          70      D0.L A0
PRINTV8D         71      D0.L A0
MULS32           72      D0.L D2.L -> D0.L D1.L
DIVS32           73      D0.L D1.L D2.L -> D0.L D1.L
FLINIT           74
FLOPPY           75      D1 D2.B D3.B D4.B A0
GETFLOP          76      D4.B -> D4.B
SETXOR           77      D0
GETXOR           78      -> D0
SETCOLOR         79      D0
GETCOLOR         80      -> D0
CURON            81
CUROFF           82
ADJ360           83      D0 -> D0
PRTSYM           84
SYMCLR           85
GETSYM           86      -> D0.L A0
GETNEXT          87      -> D0.L
PUTNEXT          88      D0
GETBASIS         89      -> D0.L A0
GETVAR           90      -> D0.L A0
SETA5            91      -> A5
AUFXY            92  (S) D1 D2
```

```
KORXY       93  (S) -> D1 D2
AUFK        94  (S) D0
GETK        95  (S) -> D0
RND         96      D0 -> D0
GETVERS     97      -> D0.L
GETSN       98      -> D0.L
CRLF        99
GETLINE     100     D0.B -> A0 A1 A2
GETCURXY    101     -> D1.L D2.L
SETCURXY    102     D1.B D2.B
GETXY       103     -> D1 D2
SI          104     -> D0.L
SO          105     D0.B
SISTS       106     -> D0.L
SOSTS       107     -> D0.L
SIINIT      108     D0.B D1.B
GETAD8      109     -> D0.L
GETAD10     110     -> D0.L
SETDA       111     D1.B D2.B
SPEAK       112     A0
SPEAK1      113     A0
SOUND       114     A0
GETUHR      115     A0
SETUHR      116     A0
LSTS        117     D0.L
RELAN       118
RELAUS      119
ASSERR      120     -> D0.L
```

C Pascal/S-Befehle

Schlüsselwörter

ARRAY	Feld definieren. ARRAY ... OF ... .
BEGIN	Anfang eines Blocks.
CASE	Fallunterscheidung.
CONST	Konstante definieren.
DIV	Ganzzahlige Division.
DO	Bei WHILE, und FOR verwendet.
DOWNTO	Rueckwärtszählen bei FOR.
ELSE	Sonst–Fall bei IF... .
END	Ende eines Blocks.

FOR	Schleife mit Zählvariable. FOR ... TO ... DO ... .
FUNCTION	Funktions-Definition.
IF	Fallunterscheidung. IF ... THEN ... ELSE ... .
MOD	Modulo-Operation.
NOT	Nicht-Verknüpfung.
OR	Oder-Verknüpfung.
PROCEDURE	Prozedur-Definition.
PROGRAM	Programm-Start.
RECORD	Definition einer zusammengesetzten Struktur.
REPEAT	Wiederholungs-Anweisung. REPEAT ... UNTIL ... .
THEN	Dann-Fall bei IF... .
TO	Aufwärtszählen bei der FOR-Schleife.
TYPE	Typen-Definition.
UNTIL	Bis-Bedingung, bei REPEAT.
VAR	Variablen-Definition.
WHILE	Solange-Schleife. WHILE ... DO ... .

Prozeduren, Funktionen und Datentypen

FALSE	Der Wert Falsch, beim Boolean-Datentyp.
TRUE	Der Wert Wahr, beim Boolean-Datentyp.
REAL	Definition einer Gleitkommazahl.
CHAR	Definition einer Zeichengröße.
BOOLEAN	Definition einer booleschen Größe.
INTEGER	Definition einer Ganzen Zahl.
ABS	Den Absolutbetrag bilden.
SQR	Das Quadrat einer Zahl bilden.
ODD	Liefert Wahr, wenn die Zahl ungerade ist.
CHR	Umwandeln in ein Zeichen.
ORD	Umwandlen in eine Ganze Zahl.
SUCC	Der Nachfolger eines Elementes.
PRED	Der Vorgänger eines Elementes.
ROUND	Aufrunden einer Zahl.
TRUNC	Abschneiden der Nachkommastellen einer Zahl.
SIN	Sinus berechnen. (Winkel in Radiant).
COS	Cosinus berechnen. (Winkel in Radiant).
EXP	Bilden der Funktion e hoch x.
LN	Der natürliche Logarithmus.
SQRT	Die Quadratwurzel.
ARCTAN	Arcustangents einer Zahl. (Ergebnis in Radiant).
EOF	Liefert Wahr, wenn das Dateiende erreicht ist.
EOLN	Liefert Wahr, wenn das Zeilende erreicht ist.
READ	Lesen von Daten.
READLN	Lesen von Daten bis zum Zeilenende.
WRITE	Schreiben von Daten.
WRITELN	Schreiben von Daten mit nachfolgendem Zeilenvorschub.

D Gosi-Befehle

Die englischen Ausdrücke sind gültig, wenn man die Option [E] als Befehl angibt, sonst werden die deutschen Wörter verwendet. Wenn nur ein Wort angegeben ist, so gilt es in beiden Einstellungen. Deutsch ist voreingestellt, wenn man keinen Optionen-Befehl gibt.

Funktionen

ASC, ASCII	Umrechnung Zeichen nach Integer.
COS	Berechnung von COS*256.
EINGABE,EG, REQUEST,RQ	Eingabe einer Zeichenkette.
KURS, HEADING	Ermitteln der aktuellen Blickrichtung.
MEM	Lesen des Inhalts einer Speicherzelle.
PORT	Lesen des Inhalts eines IO-Ports.
SIN	Berechnung von COS*256
TASTE, READCHARACTER, RC	Einlesen eines Zeichens.
TASTE?, READCHARACTER?, RC?	Feststellen ob eine Taste gedrückt ist.
XKO, XKOR	Einlesen der X-Koordinate.
YKO, YKOR	Einlesen der Y-Koordinate.
ZAHL, NUMBER	Einlesen einer Zahl.
ZZ, RANDOM	Berechnen einer Zufallszahl.

Prozeduren

AUFKURS, AK, SETHEADING, SH	Damit kann die Blickrichtung der Schildkröte eingestellt werden.
AUFX, SETX	Einstellen der X–Koordinate.
AUFY, SETY	Einstellen der Y–Koordinate.
AUFXY, SETXY	Einstellen von X– und Y–Koordinate.
BEWAHRE, SAVE	Nicht vorhanden.
BILD, DRAW	Bildschirm löschen, Schildkröte in Bildmitte.
BLINKER, CURSOR	Textcursor positionieren.
CALL	Aufruf eines Maschinenunterprogramms.
CLRINV	Löschen einer Bildseite.
DRUCKAN, TOPRINTER	Alle Text–Ausgaben auf den Drucker schalten.
DRUCKAUS, TOCRT	Alle Text–Ausgaben auf den Bildschirm.
DRUCKE, DR	Ausgabe von Texten und Variablen.
DRUCKEZEILE, DZ, PRINT, PR	Wie DRUCKE, jedoch mit nachfolgendem Zeilenvorschub.
ENDE, END	Ende einer Prozedur–Definition.
FELD, ARRAY	Dimensionieren eines Variablenfeldes.

FLIP		Automatische Bildseitenumschaltung einstellen.
LADE, READ		Nicht vorhanden.
LERNE, TO		Definieren einer Prozedur.
LINIE, LINE		Zeichnen einer Linie.
LINKS, LI, LEFT, LT		Schildkröte nach Links drehen.
LOESCHEBILD, LB, CLEARSCREEN, CS		Bildschirm löschen, Schildkröte bleibt.
LOESCHESCHIRM, LS, NODRAW, ND		Bildschirm löschen, Schildkröte ausblenden.
MEM		Speicherzelle mit einem Wert belegen.
MITTE, HOME		Schildkröte in Bildschirmmitte.
OPTION		Spezialbefehl für den Compiler.
	E	Englischer Sprachschatz ein.
	S	Stack-Option.
	M	Multiplikation, Division mit 16 Bit.
	L	Code beim Übersetzen mit ausgeben.
	G	Alles rücksetzen.
PORT		Wert an einen IO-Port ausgeben.
PRUEFE, TEST		Bedingung abfragen und merken.
RECHTS, RE, RIGHT, RT		Schildkröte nach Rechts drehen.

RUECKGABE, RG, OUTPUT, OP	Wert an eine Funktion geben.
RUECKWAERTS, RW, BACK, BK	Schildkröte schreitet einen Bildpunkt zurück.
SCHR16TEL, STEP16	Schildkröte schreitet 1/16 Bildpunkt vor.
SEITE, PAGE	Bildseite für Lesen und Schreiben anwählen.
SETZE, MAKE	Variable mit Wert belegen.
SOLANGE, SO, WHILE	Solange–Schleife.
STIFT, PEN	STIFT 0 setzt den Schreibstift schreibend, STIFT 1 setzt den Schreibstift löschend.
STIFTAB, SA, PENDOWN, PD	Schreibstift senken.
STIFTHOCH, SH, PENUP, PU	Schreibstift heben.
TEST	Nicht vorhanden.
VERGISS, FORGET	Nicht vorhanden.
VI, HT	Schildkrötensymbol unsichtbar machen.
VORWAERTS, VW, FORWARD, FD	Schildkröte schreitet einen Bildpunkt vor.

WENN,
IF — Bedingung abfragen.

WENNFALSCH,
WF,
IFFALSE, Damit wird der Merker abgefragt, der
IFF — durch PRUEFE gesetzt wurde.

WENNWAHR,
WW,
IFTRUE, — Damit wird der Merker abgefragt, der
IFT — durch PRUEFE gesetzt wurde.

WIEDERHOLE,
WH,
REPEAT — Wiederholungs-Anweisung.

ZEICHEN,
CHAR — Zeichen ausgeben.

ZEIGE,
PRINTOUT — Nicht vorhanden.

ZFELD,
STRING — Zeichenketten-Variable definieren.

ZI,
ST — Die Schildkröte wird eingeblendet.

Die 68000/68008-Befehle

ABCD.B Dx,Dy	XNZVC	Addition von BCD-Zahlen
ABCD.B -(An),-(An)	XNZVC	Addition von BCD-Zahlen
ADD.x ‹ea›,Dn	XNZVC	Binäre Addition
ADD.x Dn,‹ea›	XNZVC	Binäre Addition
ADDA.x ‹ea›,An	-	Binäre Addition mit Adressregister als Ziel
ADDI.x #konst,‹ea›	XNZVC	Binäre Addition mit Konstante als Quellop.
ADDQ.x #konst,‹ea›	XNZVC	Addiere schnell (Werte 1..8)
ADDX.x Dx,Dy	XNZVC	Addiere mit Extendbit
ADDX.x -(Ax),-(Ay)	XNZVC	Addiere mit Extendbit

AND.x ‹ea›,Dn	-NZVC	Logische Und-Verknüpfung
AND.x Dn,‹ea›	-NZVC	Logische Und-Verknüpfung
ANDI.x #konst,‹ea›	-NZVC	Logische Und-Verknüpfung mit einer Konst.
ANDI #konst,CCR	XNZVC	Und-Verknüpfung mit Condition-Code-Reg.
ANDI #konst,SR	XNZVC	Und-Verknüpfung mit Statusregister (priv)
ASL.x Dx,Dy	XNZVC	Links Schieben indirekt
ASL.x #konst,Dn	XNZVC	Links Schieben, um "konst" Bits
ASL ‹ea›	XNZVC	Links Schieben, um Eins
ASR.x Dx,Dy	XNZVC	Rechts Schieben indirekt
ASR.x #konst,Dn	XNZVC	Rechts Schieben, um "konst" Bits
ASR ‹ea›	XNZVC	Rechts Schieben, um Eins
Bcc marke	-	Bedingter Sprung, 16-Bit-Adressdistanz
Bcc.S marke	-	Bedingter Sprung, 8-Bit-Adressdistanz
BCHG Dn,‹ea›	--Z--	Prüfe ein Bit und komplementiere es
BCHG #konst,‹ea›	--Z--	Prüfe ein Bit und komplementiere es
BCLR Dn,‹ea›	--Z--	Prüfe ein Bit und setze es auf Null
BCLR #konst,‹ea›	--Z--	Prüfe ein Bit und setze es auf Null
BRA marke	-	Sprungbefehl, 16-Bit-Adressdistanz
BRA.S marke	-	Sprungbefehl, 8-Bit-Adressdistanz
BSET Dn,‹ea›	--Z--	Prüfe ein Bit und setze es auf Eins
BSET #konst,‹ea›	--Z--	Prüfe ein Bit und setze es auf Eins
BSR marke	-	Unterprogrammaufruf, 16-Bit-Adressdistanz
BSR.S marke	-	Unterprogrammaufruf, 8-Bit-Adressdistanz
BTST Dn,‹ea›	--Z--	Prüfe ein Bit
BTST #konst,‹ea›	--Z--	Prüfe ein Bit
CHK ‹ea›,Dn	-NZVC	Prüfe gegen Grenzbereich
CLR.x ‹ea›	-NZVC	Löschen den Inhalt eines Operanden
CMP.x ‹ea›,Dn	-NZVC	Vergleich zweier Operanden
CMPA.x ‹ea›,An	-NZVC	Vergleich mit einem Adressregister
CMPI.x #konst,‹ea›	-NZVC	Vergleich einer Konstanten mit Operand
CMPM.x (Ax)+,(Ay)+	-NZVC	Vergleich zweier Speicherzellen
DBcc Dn,marke	-	Prüfe Bedingung, decrementiere und spinge
DBRA Dn,marke	-	Decrementiere und springe
DIVS ‹ea›,Dn	-NZVC	Division mit Vorzeichen
DIVU ‹ea›,Dn	-NZVC	Division ohne Vorzeichen
EOR.x Dn,‹ea›	-NZVC	Logische Exklusiv-Oder-Verknüpfung
EORI.x #konst,‹ea›	-NZVC	Logische Exklusiv-Oder-Verknüpfung
EORI #konst,CCR	XNZVC	Exklusiv-Oder mit Condition-Code-Register
EORI #konst,SR	XNZVC	Exklusiv-Oder mit Status-Register (priv)
EXG Rx,Ry	-	Vertausche Registerinhalte
EXT.x Dn	-NZVC	Vorzeichenrichtige Erweiterung
ILLEGAL	-	Unzulässiger Befehl, führt zur Exception

JMP ‹ea›	-	Springe absolut
JSR ‹ea›	-	Unterprogrammaufruf absolut
LEA ‹ea›,An	-	Lade effiktive Adresse in Adressregister
LINK An,#konst	-	Rette A7 und lege neuen Stackbereich an
LSL.x Dx,Dy	XNZVC	Logische Verschiebung nach Links, indirekt
LSL.x #konst	XNZVC	Logische Verschiebung nach Links
LSL.x ‹ea›	XNZVC	Logische Verschiebung nach Links
LSR.x Dx,Dy	XNZVC	Logische Verschiebung nach Rechts, indirekt
LSR.x #konst	XNZVC	Logische Verschiebung nach Rechts
LSR.x ‹ea›	XNZVC	Logische Verschiebung nach Rechts
MOVE.x ‹ea›,‹ea›	-NZVC	Übertrage Daten von Quelle nach Ziel
MOVE ‹ea›,CCR	XNZVC	Übertrage ein Datum in Condition-Code-Reg.
MOVE ‹ea›,SR	XNZVC	Übertrage ein Datum in Status-Reg. (priv)
MOVE SR,‹ea›	XNZVC	Übertrage Status-Reg. in Zieloperand
MOVE USP,An	-	Anwenderstackpointer nach An (priv)
MOVE An,USP	-	An nach Anwenderstackpointer (priv)
MOVEA.x ‹ea›,An	-	Übertrage Wert in Adressregister
MOVEM.x reglist,‹ea›	-	Register nach Ziel speichern
MOVEM.x ‹ea›,reglist	-	Quelle nach Register
MOVEP.x Dx,d(Ay)	-	Inhalt von Dx nach Adresse d(Ay)
MOVEP.x d(Ay),Dx	-	Inhalt von Adresse d(Ay) nach Dx
MOVEQ #konst,Dn	-	Laden einer Konstanten, schnell
MULS ‹ea›,Dn	XNZVC	Multiplikation mit Vorzeichen
MULU ‹ea›,Dn	XNZVC	Multiplikation ohne Vorzeichen
NBCD ‹ea›	XNZVC	Negation von BCD-Zahlen
NEG.x ‹ea›	XNZVC	Negation
NEGX.x ‹ea›	XNZVC	Negation mit Berücksichtigung des X-Flags
NOP	-	Keine Operation (Nichtstubefehl)
NOT.x ‹ea›	XNZVC	Logisches Komplement
OR.x ‹ea›,Dn	-NZVC	Logische Oder-Verknüpfung
OR.x Dn,‹ea›	-NZVC	Logische Oder-Verknüpfung
ORI.x #konst,‹ea›	-NZVC	Logische Oder-Verknüpfung mit Konstanten
ORI #konst,CCR	XNZVC	Logisches Oder mit Condition-Code-Register
ORI #konst,SR	XNZVC	Logisches Oder mit Status-Register (priv)
PEA ‹ea›	-	Lege die effektive Adresse auf dem Stack ab
RESET	-	Rücksetzen externen Peripherie (priv)
ROL.x Dx,Dy	XNZVC	Rotiere links, indirekt
ROL.x #konst,Dy	XNZVC	Rotiere links, um Anzahl "konst"
ROL.x ‹ea›	XNZVC	Rotiere rechts, um Eins
ROR.x Dx,Dy	XNZVC	Rotiere rechts, indirekt
ROR.x #konst,Dy	XNZVC	Rotiere rechts, um Anzahl "konst"
ROR.x ‹ea›	XNZVC	Rotiere rechts, um Eins

ROXL.x Dx,Dy	XNZVC	Rotiere links, indirekt mit X-Flag
ROXL.x #konst,Dy	XNZVC	Rotiere links, um Anzahl "konst" mit X-Flag
ROXL.x ‹ea›	XNZVC	Rotiere rechts, um Eins mit X-Flag
ROXR.x Dx,Dy	XNZVC	Rotiere rechts, indirekt mit X-Flag
ROXR.x #konst,Dy	XNZVC	Rotiere rechts, um Anzahl "konst" mit X-Flag
ROXR.x ‹ea›	XNZVC	Rotiere rechts, um Eins mit X-Flag
RTE	XNZVC	Rückkehr aus Exception (priv)
RTR	XNZVC	Rückkehr mit Laden der Flags
RTS	-	Rückkehr aus dem Unterprogramm
SBCD Dx,Dy	XNZVC	Subtraktion von BCD-Zahlen
SBCD -(Ax),-(Ay)	XNZVC	Subtraktion von BCD-Zahlen
Scc	XNZVC	Setze ein Byte je nach Bedingung
STOP #konst	XNZVC	Lade Statusregister und HALT (priv)
SUB.x ‹ea›,Dn	XNZVC	Binäre Subtraktion
SUB.x Dn,‹ea›	XNZVC	Binäre Subtraktion
SUBA.x ‹ea›,An	-	Binäre Subtraktion ins Adressregister
SUBI.x #konst,‹ea›	XNZVC	Binäre Subrtraktion mit einer Konstanten
SUBQ.x #konst,‹ea›	XNZVC	Subtrahiere schnelle (1..8)
SUBX.x Dx,Dy	XNZVC	Subtrahiere mit Extendbit
SUBX.x -(Ax),-(Ay)	XNZVC	Subtrahiere mit Extendbit
SWAP Dn	-NZVC	Tausche Inhalt von Registerhälften
TAS ‹ea›	-NZVC	Prüfe und setze ein Bit im Zieloperanden
TRAP #konst	-	Trap-Befehl
TRAPV	-	Trap bei Überlauf
TST.x ‹ea›	-NZVC	Prüfe den Inhalt eines Operanden
UNLK An	-	Hole den Stackpointer zurück

F Folgende Medien beschäftigen sich mit dem NDR-KLEIN-Computer:

1. Hersteller der Baugruppen.
   Die Firma GES (siehe Bezugsquellenverzeichnis) liefert alle notwendigen Bausätze zum NDR-KLEIN-Computer. Ebenfalls sind aktuelle Aufbauanleitungen zu den Bausätzen verfügbar, diese sind auch einzeln erhältlich.

2. Die Benutzerzeitschrift LOOP.
   Sie liefert aktuelle Beispiele zum NDR-KLEIN-Computer und wird ebenfalls von GES geliefert.

3. Franzis-Software-Service.
   Geliefert werden alle EPROMs zum NDR-KLEIN-Computer, ferner die Programmlistings in gebundener Form, und Programme auf Disketten.

4. Christiani-Lehrgänge.
   Christiani bietet Lehrgänge zum NDR-KLEIN-Computer an, mit denen man den Stoff systematisch erlernen kann.

5. Fernsehsendung.
   Die Sendungen „Rechner modular" und „Mikroelektronik" bieten Grundlagen und Anregungen zum NDR-KLEIN-Computer; sie führen in das Programmieren ein.

6. Sonderhefte.
   Im Hause Franzis gibt es die beiden Sonderhefte „Mikrocomputer Schritt für Schritt" 1 und 2, die eine Einführung zum NDR-KLEIN-Computer bieten.

7. Franzis-Bücher zum NDR-KLEIN-Computer:
   Klein, Mikrocomputer selbstgebaut und programmiert;
   Klein, Mit HEXMON Programme entwickeln.

Über weitere Bücher und sonstige Unterlagen fragen Sie bitte aktuell beim Franzis-Verlag nach.

G Schemas und Kurzbeschreibung der verwendeten ICs

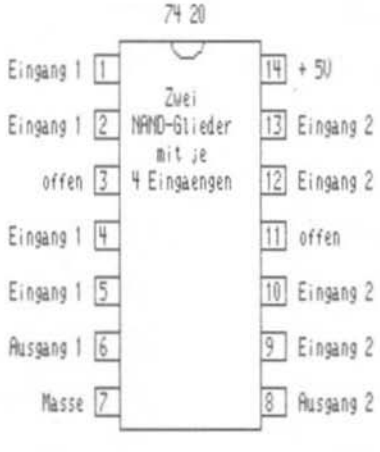
74 00

74 04

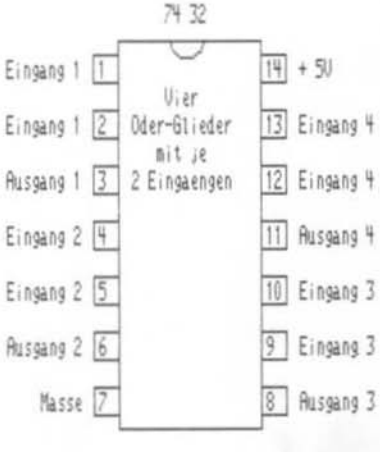
74 05 / 74 06

74 LS 14

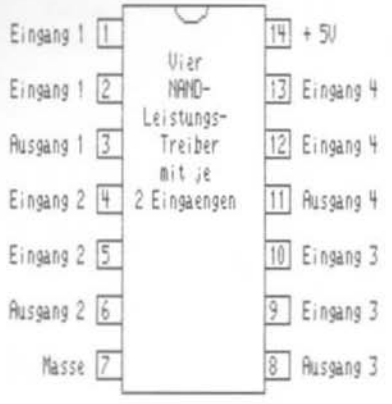
74 20



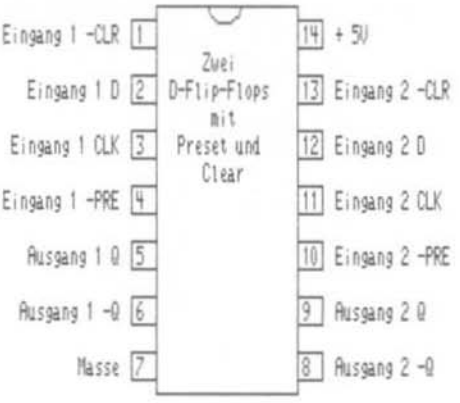
74 32



74 38

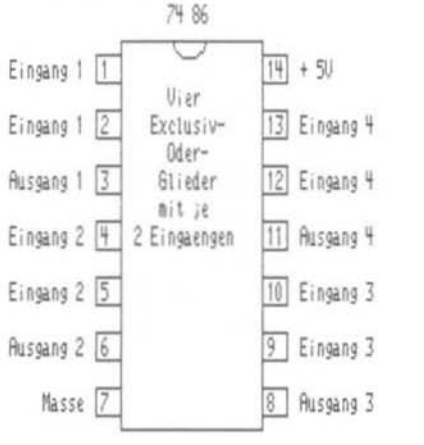


74 74

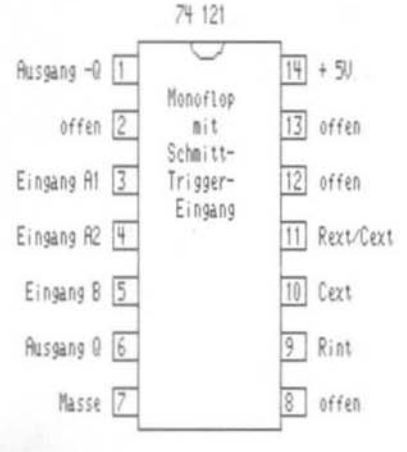


74 85

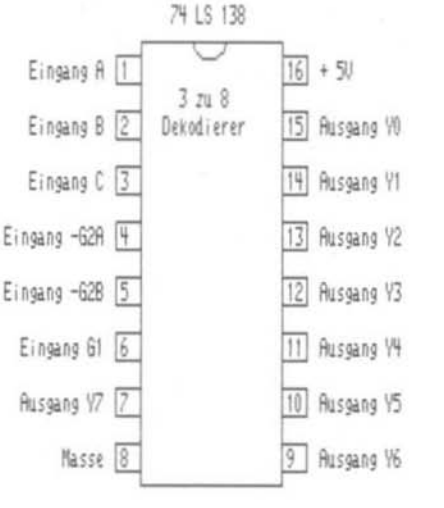
74 86



74 121



74 LS 138



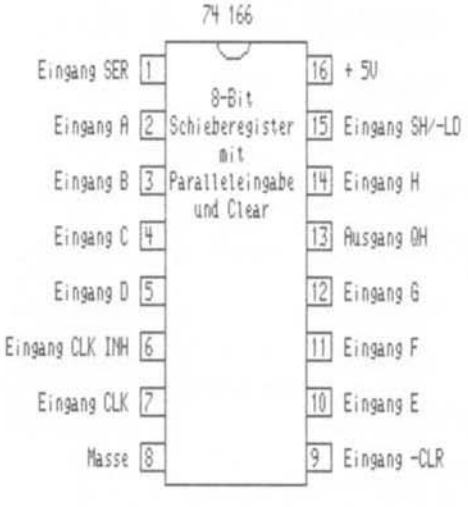
74 LS 139

74 153

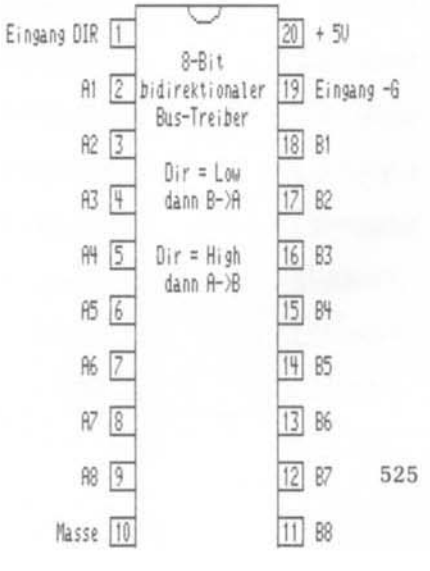
74 163

74 164

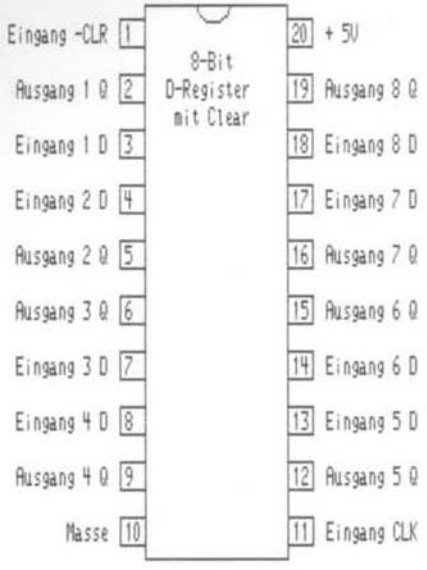
74 166



74 LS 245

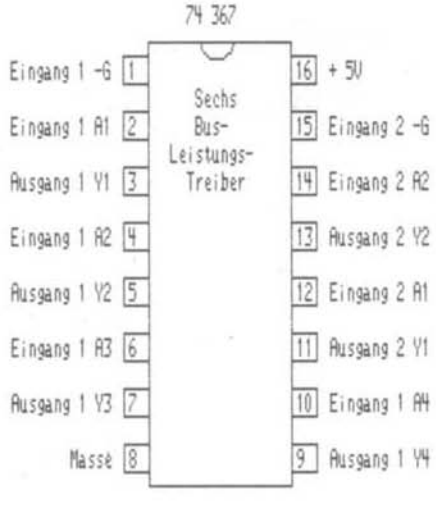


74 273

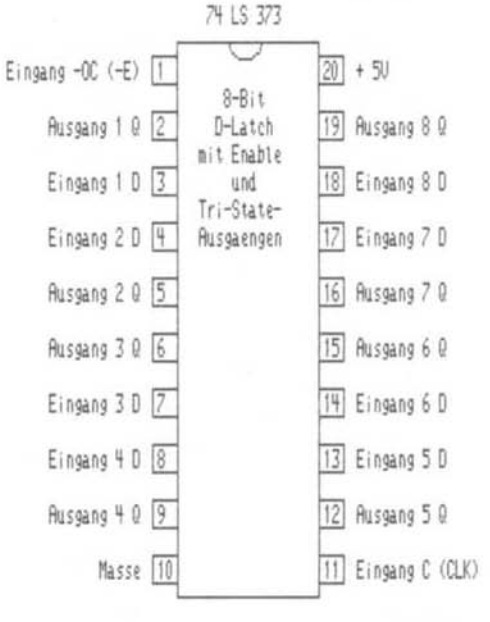


74 LS 244

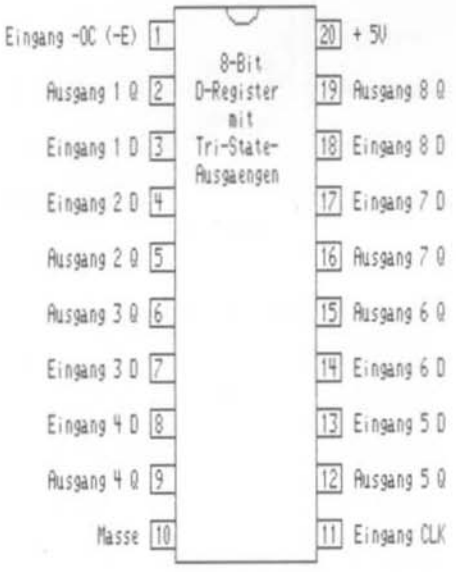
74 367



74 LS 373

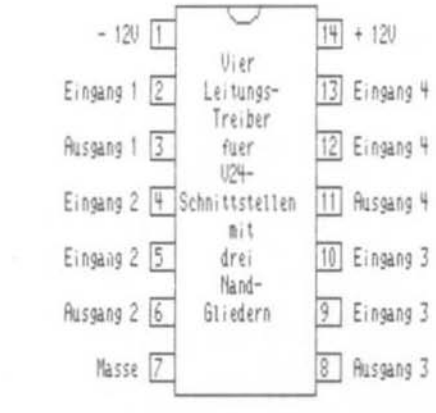


74 LS 374



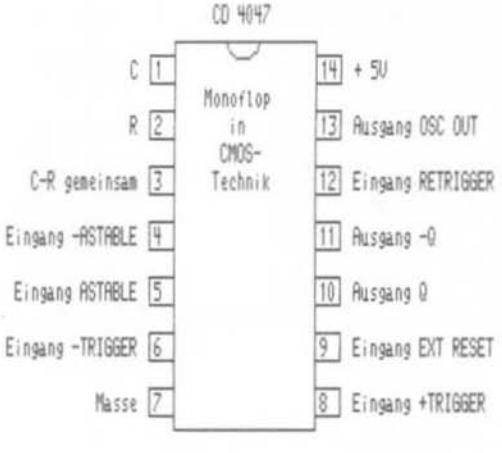
75 189

75 188

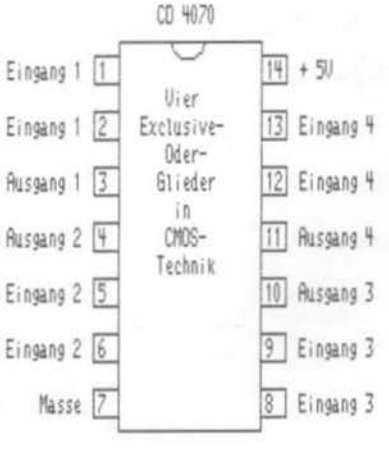


CA 3130

CD 4047

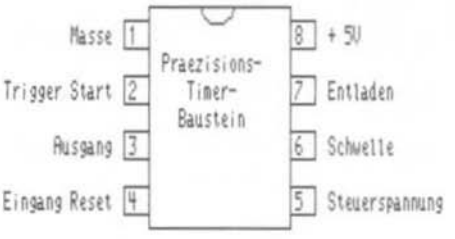


CD 4070

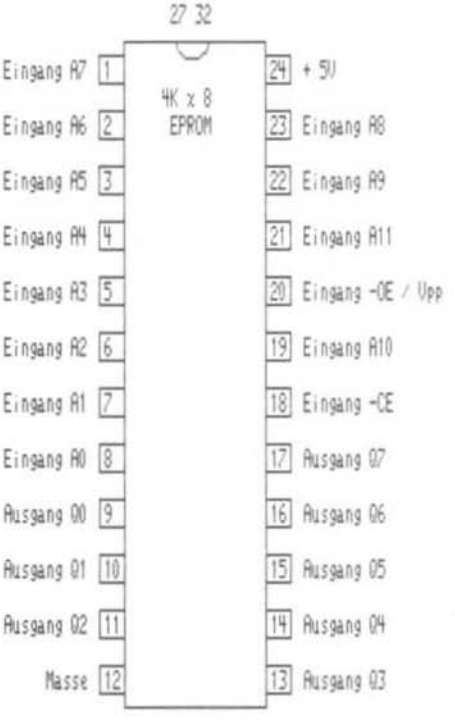


CD 4528

NE 555

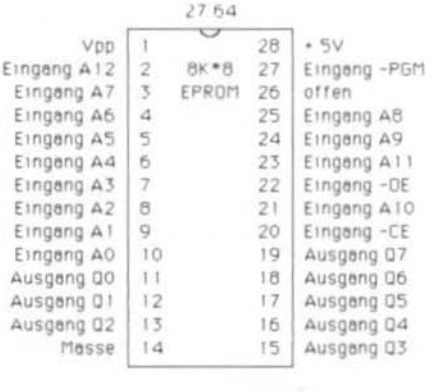


27 32

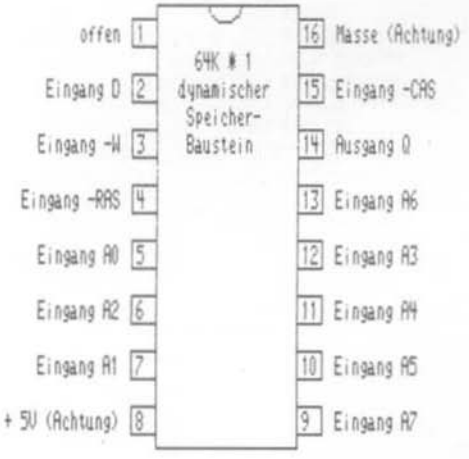


27 16

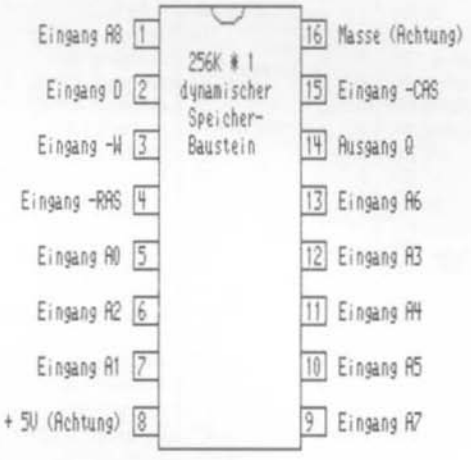
27 64



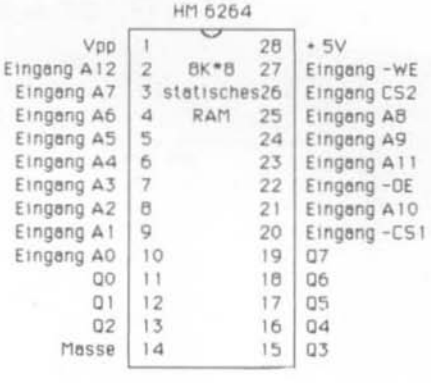
8264 / 4164 usw.



81256 / 41256 usw.



HM 6264



HM 6116

EF 9365/66

FD 1797

offen	1	40	+ 12 V
Eingang -WE	2	39	Ausgang INTRQ
Eingang -CS	3	38	Ausgang DRQ
Eingang -RE	4	37	Eingang -DDEN
Eingang A0	5	36	Eingang -WPRT
Eingang A1	6	35	Eingang -IP
D0	7	34	Eingang -TR00
D1	8	33	Eingang -WF/-VFOE
D2	9	32	Eingang READY
D3	10	31	Ausgang WD
D4	11	30	Ausgang WG
D5	12	29	Ausgang TG43
D6	13	28	Ausgang HLD
D7	14	27	Eingang -RAW READ
Ausgang STEP	15	26	Eingang RCLK
Ausgang DIRC	16	25	Ausgang SSD
Ausgang EARLY	17	24	Eingang CLK
Ausgang LATE	18	23	Eingang HLT
Eingang -MR	19	22	Eingang -TEST
Masse	20	21	+ 5V

Floppy-Disk-Controller

SY 6551

Masse	1	28	Eingang R/-W
Eingang CS0	2	27	Eingang ø2
Eingang -CS1	3	26	Ausgang -IRQ
Eingang -RES	4	25	D7
RxC	5	24	D6
XTAL 1	6	23	D5
XTAL 2	7	22	D4
Ausgang -RTS	8	21	D3
Eingang -CTS	9	20	D2
Ausgang TxD	10	19	D1
Ausgang -DTR	11	18	D0
Eingang RxD	12	17	Eingang -DSR
Eingang RS0	13	16	Eingang -DCD
Eingang RS1	14	15	+ 5V

Serieller Interface-Baustein

68000

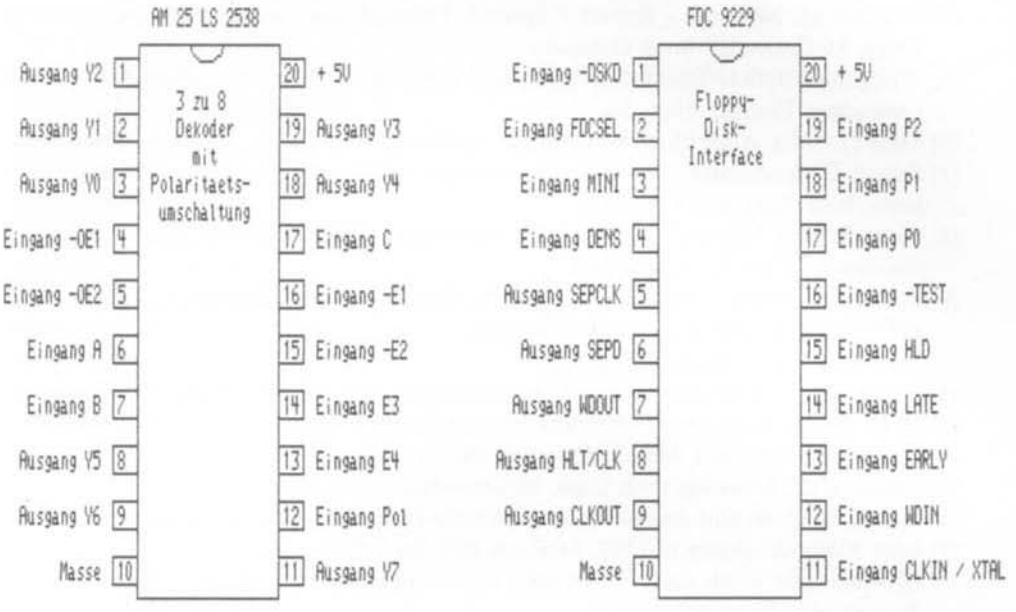
D4	1	64	D5
D3	2	63	D6
D2	3	62	D7
D1	4	61	D8
D0	5	60	D9
Ausgang -AS	6	59	D10
Ausgang -UDS	7	58	D11
Ausgang -LDS	8	57	D12
Ausgang R/-W	9	56	D13
Eingang -DTACK	10	55	D14
Ausgang -BG	11	54	D15
Eingang -BGACK	12	53	Masse
Eingang -BR	13	52	Ausgang A23
+ 5V	14	51	Ausgang A22
Eingang CLK	15	50	Ausgang A21
Masse	16	49	+ 5V
-HALT	17	48	Ausgang A20
-RESET	18	47	Ausgang A19
Ausgang -VMA	19	46	Ausgang A18
Ausgang E	20	45	Ausgang A17
Eingang -VPA	21	44	Ausgang A16
Eingang -BERR	22	43	Ausgang A15
Eingang -IPL2	23	42	Ausgang A14
Eingang -IPL1	24	41	Ausgang A13
Eingang -IPL0	25	40	Ausgang A12
Ausgang FC2	26	39	Ausgang A11
Ausgang FC1	27	38	Ausgang A10
Ausgang FC0	28	37	Ausgang A9
Ausgang A1	29	36	Ausgang A8
Ausgang A2	30	35	Ausgang A7
Ausgang A3	31	34	Ausgang A6
Ausgang A4	32	33	Ausgang A5

CPU

68008

Ausgang A3	1	48	Ausgang A2
Ausgang A4	2	47	Ausgang A1
Ausgang A5	3	46	Ausgang A0
Ausgang A6	4	45	Ausgang FC0
Ausgang A7	5	44	Ausgang FC1
Ausgang A8	6	43	Ausgang FC2
Ausgang A9	7	42	Eingang -IPL2/-IPL0
Ausgang A10	8	41	Eingang -IPL1
Ausgang A11	9	40	Eingang -BERR
Ausgang A12	10	39	Eingang -VPA
Ausgang A13	11	38	Ausgang E
Ausgang A14	12	37	-RESET
+ 5V	13	36	-HALT
Ausgang A15	14	35	Masse
Masse	15	34	Eingang CLK
Ausgang A16	16	33	Eingang -BR
Ausgang A17	17	32	Ausgang -BG
Ausgang A18	18	31	Eingang -DTACK
Ausgang A19	19	30	Ausgang R/-W
D7	20	29	Ausgang -DS
D6	21	28	Ausgang -AS
D5	22	27	D0
D4	23	26	D1
D3	24	25	D2

CPU



H Literaturverzeichnis

Allgemein, bzw. zum 68000/68008:

[1] Werner Hilf - Anton Nausch. M68000 Familie Teil 1: Grundlagen und Architektur. Te-wi

[2] Werner Hilf - Anton Nausch. M68000 Familie Teil 2: Anwendung und 68000 Bausteine. Te-wi

[3] 16-Bit Microprozessor Users Manual. Motorola.

[4] Klein, Rolf-Dieter. Mikrocomputer selbstgebaut und programmiert. Farnzis-Verlag.

[5] J. Plate. Betriebsystem CP/M. Franzis-Verlag.

[6] Klein, Rolf-Dieter. Mit HEXMON Programme entwickeln. Franzis-Verlag.

[7] Gerry Kane...Leventhal, 68000 Assembly Language Programming. Osborne/ McGraw-Hill.

zum Thema Graphik:

[1] Günter Pomaska, Computergraphik 2D- und 3D-Programmierung, Vogel-Buchverlag.

[2] J. D. Foley - A.van Dam, Fundamentals of Interactive Computer Graphics, Addison-Wesley Publishing Company.

[3] Computer Graphics - Theory and Applications, Springer-Verlag, 1983.

[4] William M. Newman - Robert F.Sproull, Prinziples of Interactive Computer Graphics, McGraw-Hill Book Company.
[5] Wolfgang K.Giloi, Interactive Computer Graphics - Datß Structures, Algorithms, Languages, Prentice-Hall, Inc.
[6] Theo Pavlidis, Algorithms for Graphics and Image Processing, Springer-Verlag 1982.
[7] Benoit B.Mandelbrot, The Fractal Geometrie of Nature, W.H. Freeman and Company, New York-San Fransisco
[8] Faber-Brixius, Lineare Algebrß und Analytische Geometrie - Vektorräume und affine Geometrie, Ernst Klett Stuttgart.
[9] Köhler-Höwelmann-Krämer, Analytische Geometrie und Abbildungsgeometrie und vektorieller Darstellung. Diesterweg Salle.

zum Thema Logo und Lisp:

[1] Harald Abelson, Einführung in Logo, (übersetzt von Herbert Löthe), IWT-Verlag.
[2] H. U. Hoppe, Logo im Mathematik-Unterricht, IWT-Verlag.
[3] G. Gärtner, Logo mit dem Commodore 64, IWT-Verlag.
[4] Daniel Watt, Learning with Logo, McGraw-Hill Book Company.
[5] Harold Abelson and Andreß diSessa, Turtle Geometry, The MIT Press.
[6] John Allen, Anatomy of LISP, McGraw-Hill Book Company.

Hinweis: Das Buch nach [5] ist sehr ausführlich und enthält auch tieferführende Aspekte der Sprache Logo.
Die Bücher nach [1], [2], [3] verwenden das deutschsprachige Logo, daher kann man Beispiele leicht in Gosi übertragen.

zum Thema Pascal:

[1] Kathleen Jensen, Niklaus Wirth, Pascal User Manual and Report, Springer-Verlag.
[2] Peter Grogono, Programming in Pascal, Addison-Wesley.
[3] Kernighan - Plauger, Software Tools in Pascal, Addison-Wesley.
[4] N. Wirth. Systematisches Programmieren, Teubner Studienbücher.
[5] N. Wirth. Algorithem und Datenstrukturen, Teubner Studienbücher.
[6] N. With. Compilerbau, teubner Studienbücher.
[7] Klein, Rolf-Dieter. Was ist Pascal. Franzis-Verlag.
[8] J. Plate, Pascal: Einführung-Programmentwicklung-Strukturen, Franzis-Verlag.
[9] Busch. Der sichere Einstieg in Pascal. Franzis-Verlag.

Hinweis: im Buch [9] wird ausschließlich das Pascal/S mit dem NDR-KLEIN-Computer verwendet. Alle Programme sind damit direkt einsetzbar.

Sonderdrucke:

Über aktuelle Sonderdrucke zum NDR-KLEIN-Computer informieren Sie sich beim Franzis-Verlag, Abteilung Software-Service.

I Bezugsquellenverzeichnis

Original-Hersteller der Leiterplatten und Bausätze (auch EPROMs etc.):
Graf-Elektronik Systeme GmbH
Magnusstraße 13
Postfach 1610
8968 Kempten
Tel. 0831/6211 oder 0831/61930
Mailbox: 0831/69330
Software: EPROMs Disketten und Literatur:
Franzis-Verlag
Software-Service
Karlstraße 37
8000 München 37
Tel: 089/ 5117-331

J Terminologieverzeichnis

A

Access
Zugriff; Zugriff zum Beispiel auf einen Speicher.

Ada
Eine Programmiersprache, die im Auftrag des amerikanischen Verteidigungsministerium entwickelt wurde. Ada ist eine sehr umfangreiche Programmiersprache, die auch Multitasking-Konzepte usw. enthält.
Beispiel:

```
loop
  select
   accept  READ( X: out ITEM) do
     X := STORED ;
   end READ ;
  or
   accept  WRITE(X : ITEM) do
     STORED := X ;
   end WRITE ;
  end select ;
end loop ;
```

Adresse
Eine Bezeichnung für einen bestimmten Speicherplatz oder Speicherbereich.

Adreßbus
Ein Bus auf den die Adreßleitungen eines Mikroprozessors geführt werden.

Akkumulator
Ein Register, in dem arithmetische und logische Operationen ausgeführt werden können.

Algol
Algorithmic Language. Es handelt sich um eine Programmiersprache für den technisch wissenschaftlichen Bereich.
Programmbeispiel:

```
'BEGIN'
  'REAL' alpha, betha;
  alpha := 3.1;
  INREAL(1,betha);
  OUTREAL(2,betha*alpha);
'END'
```

ALU
Arithmetic Logic Unit, Rechenwerk. In diesem Teil des Rechners werden die arithmetischen und logischen Verknüpfungen ausgeführt.

APL
A Programming Language. Eine Programmiersprache für den technisch wissenschaftlichen Bereich, die Sprache verwendet dabei spezielle Zeichen. Beispiel:

```
    ∇ R←X   WL   Y
[1]   ⍝ Waagrechte Linie einfuegen
[2]   R←(~R ∈ X)/R←↑(⌊1↑⍴Y),X
[3]   R←(Y,[⎕IO]' ')[(⎕IO+1↑⍴Y)⌊R;]
    ∇
```

APU
Arithmetic Processor Unit, siehe auch FPU

ASCII
American Standard Code for Information Interchange. Wird auch mit ISO-7-Bit-Code bezeichnet (DIN 66003). Codierungsart für Zeichen.

Assembler
Ein Übersetzungsprogramm, das aus einem mnemotechnischen Programm-Code einen Maschinen-Code erstellt

B

Bankselekt
Unter einer Speicher-Bank versteht man zum Beispiel eine Gruppe von Speichern. Selekt steht für Auswahl. Bankselekt bedeutet also die Auswahl einer Gruppe von Speichern. Man verwendet ein Bankselekt-Signal zum Beispiel zum Erweitern des Adreßraums

BAS-Mischer
Aus dem Synchron- und Videosignal wird durch elektrisches Mischen ein Signal gewonnen, das beide Signale auf einer Leitung transportieren kann. Dieses BAS-Signal ist zur Ansteuerung von vielen Monitoren geeignet.

Basic
Beginners All Purpose Symbolic Instruction Code. Eine einfache Programmiersprache, die besonders auf Heimcomputern sehr verbreitet ist, jedoch den Nachteil besitzt, nicht strukturiert zu sein.
Beispiel:

```
10 PRINT "Quadratwurzeltabelle"
20 FOR I=1 TO 10
30 PRINT I,SQRT(I)
40 NEXT I
```

Baudrate
Messung des Datenflusses, wobei die Zeit zur Übertragung des kürzesten Elements als Maß genommen wird. Beispiel: 1200 Baud bedeuten eine Übertragung von 1200 Bit pro Sekunde

Baudrate-Generator
Ein Baustein zur Erzeugung eines Taktes, der dann für eine serielle Übertragung verwendet wird.

Betriebssystem
Eine Reihe von Programmen, die es dem Computer ermöglichen, selbstständig Programme zu bearbeiten, CP/M und MSDOS sind z.B. solche Betriebssysteme für Mikrorechner.

Bi-Direktionale Bustreiber
Ein Schaltkreis, der logische Informationen auf einer Leitung in beiden Richtungen übertragen kann.

Bildwiederholspeicher
Die Information, zur Darstellung auf dem Bildschirm wird im Bildwiederholspeicher bereit gehalten, so daß sie fortlaufend ausgegeben werden kann.

Bit
Binary Digit. Kleinste Informationseinheit

Boot
Das Neustarten eines Systems, bei dem Programme geladen werden, wird auch als Boot bezeichnet.

Branch
Verzweigung, Sprung

Buffer
Puffer; Speicher in dem Daten kurzzeitig festgehalten werden, oder auch Treiber zum Schalten von größeren Lasten.

Bus
Sammelleitung, an die mehrere Bausteine angeschlossen werden können. Dabei können auch mehrere Bausteine Daten auf den Bus angegeben, jedoch nicht zur gleichen Zeit. Eine Auswahllogik sorgt dafür, daß immer nur ein Busteilnehmer sendet, wobei jedoch alle weiteren hören dürfen.

Byte
8 Bits werden als 1 Byte zusammengefaßt

C
Eine Programmiersprache, die zum Beispiel mit dem CP/M68k-Betriebssystem geliefert wird. Die Sprache ähnelt sehr der Sprache Pascal. Programmbeispiel:

```
main()
{
  int i;
  float sqrt();
  for (i = 1; i<=10; i++) {
    printf(
      " Wurzel aus %d ist   %f",
      i,sqrt(i));
  }
}
```

Cache
Ein schneller Speicher, der z.B.in dem Prozessor-IC integriert ist, und es z.B. erlaubt kleine Programmschleifen schnell ausführen zu können. Der Prozessor 68020 besitzt z.B. einen Programm-Cache.

Clock
Takt

Cobol
Common Business Oriented Language. Eine Programmiersprache vorwiegend für kaufmännische Probleme. Beispiel:

```
PROCEDURE DIVISION.
START.
  MOVE ZEROS TO N
  MOVE 1 TO FAKULTAET.
  ACCEPT M-EIN FROM
              KARTEN-LESER;
  MOVE M-EIN TO M
  ...
```

Comal
Common Algorithmic Language; diese Sprache entstand 1973 aus einer Mischung von Basic und Pascal. Sie enthält daher strukturierte Elemente und Parametermechanismen
Programmbeispiel:

```
0010 PROC FENSTER(X,Y) CLOSED
0020   DIM LEERZ$ OF 40
0030   LEERZ$(1:40) := " "
0040   POSI(X,1)
0050   FOR ZN:=1 TO Y-X+1 DO PRINT
         LEERZ$
0060   POSI(X,1)
0070 ENDPROC FENSTER
```

Compiler
Ein Übersetzungsprogramm, das eine höhere Programmiersprache in den Maschinen-Code übersetzt. Siehe Kapitel Pascal/S und Gosi

Conditional
Bedingt

Controller
Steuereinheit

CP/M
Disk Operating System für 8080, 8085, Z80, 8086 und 68000 von DIGITAL RESEARCH. Siehe Kapitel Betriebssysteme

Cross-Assembler
Ein Assembler, der nicht auf der Maschine läuft, für die er Code erzeugt. Zum Beispiel ist ein Assembler für den 68000 ein Cross-Assembler, wenn man ihn unter CP/M80, also z.B. auf dem Z80 laufen lassen kann.

Cross-Compiler
Ein Übersetzer für eine höhere Programmiersprache, der auf einem anderen Prozessor läuft, als für welchen er Maschinen-Code erzeugt.

CRT
Cathode Ray Tube; Datensichtgerät oder Bildschirm

Cursor
Sichtmarke zur Kennzeichnung der aktuellen Schreibposition auf dem Bildschirm.

D

Datenbus
Ein Bus, auf den die Datenleitungen eines Mikroprozessors geführt werden.
Debugging
Wörtlich "entwanzen". Gemeint ist die Fehlersuche und Beseitigung in Programmen.
Decrement
Erniedrigen, herunterzählen
Digit
Ziffer, Stelle
DIL
Dual In Line - Gehäuseform
Direktory
Inhaltsverzeichnis: z.B. von einer Diskette
DMA
Direct Memory Access; direkter Zugriff auf den Speicher eines Rechners, wobei die Zugriffssteuerung von einer Peripherieeinheit vorgenommen wird.
DOS
Disc Operating System, Betriebssystem; siehe Kapitel Betriebssystem
dynamische Speicher
Ein Speicher, bei dem die Speicherzellen ständig angesprochen werden müssen, damit sie ihre Information nicht verlieren.

E

Editor
Ein Programm, das die Eingabe von Text erlaubt.
EEPROM
Electrical Erasable Programmable Read Only Memory. Ein Speicher, der sich elektrisch programmieren und löschen läßt. Dabei bleibt die Information nach dem Ausschalten der Versorgungsspannung erhalten. Im Gegensatz zu den EPROMs werden die EEPROMs durch Anlegen einer höheren Spannung gelöscht.
Emulation
Softwaremäßige Nachbildung eines Computers, so daß der Befehlssatz des einen Computers auf einem anderen verfügbar wird. Für den 68000/8 gibt es einen Z80-Emulator, so daß man Z80 Programme ablaufen lassen kann, obwohl man einen 68000/8 verwendet.
Enable
Freigabe
enter
Eingeben
EPROM
Erasable Programmable Read Only Memory. Ein löschbarer Nur-Lese-Speicher. Siehe Kapitel PROMMER
erase
Löschen
Error
Fehler, Irrtum
Even
bedeutet gerade im Gegensatz zu ungerade
Expression
Ausdruck

Fan-in
Eingangslastfaktor
Fan-out
Ausgangslastfaktor; er gibt an, wieviele Bausteine der gleichen Logikserie an diesem Ausgang angeschlossen werden dürfen.
Festwertspeicher
Ein Speicher, dessen Inhalt nicht (oder nur mit Mühe) geändert werden kann.
Fifo
First In First Out. Zuerst eingehende Daten werden auch zuerst wieder ausgegeben.
File
Datei, Daten. Eine Ansammlung von Datengruppen, die in einer Datei angelegt werden.
Firmware
Eine Software, die fest zur Funktionsfähigkeit eines Systems nötig ist und z.B. in einem ROM abgelegt ist.
Fixed-Point
Festkomma
Flag
Eine Marke oder ein Flip-Flop zum Festhalten eines Zustands.
Floating-Point
Gleitkomma
Forth
Eine Programmiersprache, die auf der UPN (umgekehrt polnische Notation) basiert. Beispiel:

```
: TEXTAUS ."Hallo Forth Erg="
  + * . ;
3 4 5 TEXTAUS
```

Fortran
Formula Translation. Eine problemorientierte Programmiersprache für den technisch wissenschaftlichen Bereich. Beispiel:

```
      SUBPROGRAM BEISPIEL
       COMPLEX Z1,Z2
C     komplexe Zahlen sind möglich
      READ* A,B,C
      D = (B*B-4.*A*C)/(4.*A*A)
      IF (D.GE.0.) GOTO 20
      Z1 = COMPLX(-B/2.*A),SQRT(-D)
      Z2 = CONJG(Z1)
      PRINT*, 'Z1=',Z1,' Z2=',Z2
20    STOP
      END
```

FPU
Floating Point Unit. Gleitkommarechner. Für den 68020 und 68000/8 gibt es z.B. den Baustein 68881, der eine FPU darstellt.
FSK
Frequency shift. Verfahren bei der Aufzeichnung auf Datenträger.

G

Gate
Verknüpfungsschaltung
GND
Masseanschluß, 0V

H

Handshake
Quittungsbetrieb. Durch Steuersignale werden Geräte mit verschiedenen Arbeitsgeschwindigkeiten synchronisiert.
Hardcopy
Kopie. Zum Beispiel Ausdruck eines Bildschirminhalts
Hardware
Damit sind alle Bauteile, Geräte eines Systems gemeint.
Hexadezimal
Siehe Sedezimal
High order
Höherwertige Stelle

I

ICE
In-Circuit Emulator. Gerät zum Test und Entwicklung von Mikrorechnerschaltungen
Increment
Erhöhen, raufzählen
Initialisierung
Die Anfangsschritte in einem Programm, um definierte Startwerte zu erhalten
Input
Eingabe
Instruktionszyklus
Ablauf eines Befehlsausführungsvorgangs
INT
Interrupt. Unterbrechungsanforderung
Interpreter
z.B. ein Programm, das Befehle einer höheren Programmiersprache direkt ausführt und sie nicht vorher in Maschinensprache übersetzt.
Interrupt
Unterbrechung

J

Job
Auftrag
Joystick
Entweder ein Steuerknüppel mit vier Kontakten in den Endstellungen, oder ein Steuerknüppel mit zwei Potentiometern
Jump
Sprung

K

Keyboard
Tastatur
Kit
Bausatz
kompatibel
Austauschbar, aneinander angepaßt;

L

Label
Marke. In Programmiersprachen ist damit meist eine symbolische Adresse gemeint
Lichtgriffel
Ein Stift mit einem optischen Aufnehmer, der auf den Bildschirm gehalten wird. Der Rechner kann dann die Position des Lichtgriffels ermitteln.
Lifo
Last In First Out. Zuletzt gespeicherte Daten werden zuerst ausgegeben (Stack).
Linker
Ein Programm, das mehrere Teilprogramme, die schon übersetzt wurden zu einem gesamten Programm zusammenfügen kann. Dabei können die Teilprogramme Bezüge untereinander enthalten
Lisp
List Processing. Eine Programmiersprache für die Verarbeitung von Listenstrukturen und rekursiver Technik für Probleme der künstlichen Intelligenz. Beispiel:

```
(DEFLIST
 ((CAAR(LAMBDA(X)(CAR(CAR X))))
  (CADR(LAMBDA(X)(CAR(CDR X))))
  (CDAR(LAMBDA(X)(CDR(CAR X))))
  (CDDR(LAMBDA(X)(CDR(CDR X)))) )
 EXPR )
```

Listing
Ausdruck, Auflistung

Loader
Ein Ladeprogramm
Logik Analysator
Ein Geräte, mit dem man digitale Schaltungen auf ihr Zeitverhalten untersuchen kann. Ferner gibt es für Mikroprozessoren auch die Möglichkeit den Befehlsablauf sichtbar zu machen.
Logo
Eine Programmiersprache, die ähnlich wie Lisp auch mit Listen arbeitet, jedoch zusätzlich graphische Ausgabebefehle besitzt. Die Sprache wurde zum Programmieren und Lernen für Kinder entwickelt, bietet jedoch Fähigkeiten, die bis in den Hochschulbereich reichen. Die Sprache gibt es für unterschiedliche Nationalitäten. Eine Besonderheit ist, daß man sich selbst Befehle definieren kann, die dann die Sprache erweitern.
Beispiel:

```
LERNE #ÜBERSETZE :L
 WENN :L = [ ] DANN RÜCKKEHR
  DEF ERSTES :L #UEB PRLISTE
                      ERSTES :L
 #ÜBERSETZE OHNEERSTES :L
ENDE
LERNE KREIS :N
 WH 360 [ VW :N RE 1]
ENDE
```

Loop
Schleife; durch einen Sprung zurück kann eine Programmschleife entstehen.
Low order
Niederwertige Stelle

M

Mark
Bei der seriellen Übertragungs ist damit der logische Wert 1 gemeint, im Gegensatz zu Space.
Maschinenbefehl
Ein Befehl, den der Computer unmittelbar verstehen kann.
maskieren
Damit kann die Ausführung von Interrupts z.B. gestoppt oder freigegeben werden. Ferner bedeutet maskieren auch bestimmte Bits ausblenden.
Memory
Speicher
Mikrocomputer
Besteht aus einem Mikroprozessor, Speichern und Peripherie.
Mikroprogrammierbar
Der Befehlssatz eines Prozessors kann mit Hilfe von Mikrobefehlen definiert werden. Nicht mit Maschinensprache zu verwechseln. Der 68000/8 enthält z.B. ein Mikroprogramm das in seinem Inneren abgelegt ist und den Befehlssatz definiert, es kann jedoch nicht verändert werden.
Mikroprozessor
Ein integrierter Baustein, als Teil eines Mikrocomputers, der ein Leit- und Rechenwerk besitzt.
Mikrorechner
Ein Computer, der mit einem Mikroprozessor aufgebaut ist.
mnemotechnische Darstellung
Leicht zu merkende Abkürzungen für längere Begriffe, z.B. ist BRA die mnemotechnische Abkürzung für Branch.
Modem
Modulator und Demodulator. Eine Schaltung, die Daten für eine Fernübertragung aufbereitet. Ein Akkustikkoppler ist zum Beispiel ein solches Modem.
Modula 2
Eine Programmiersprache, die alle Konzepte von Pascal enthält, zusätzlich jedoch das Modul-Konzept beinhaltet.
Beispiel:

```
DEFINITION MODULE Buffer;
  EXPORT QUALIFIED ablegen, holen,
    nichtleer, nichtvoll;
  VAR nichtleer, nichtvoll : BOOLEAN;
  PROCEDURE ablegen(x : CARDINAL);
  PROCEDURE abholen(VAR x :
                        CARDINAL);
END Buffer.
```

Monoflop
Ein bistabiles Speicherelement, das nach dem Auslösen nur eine bestimmte Zeit in dem neuen Zustand bleibt und danach wieder in den Ruhezustand zurückfällt.
Multiplex
Übertragung von mehreren verschiedenen Informationen, die dazu zeitlich hintereinander übertragen werden.
Multiprozessing
Ein aus mehreren CPUs oder Teil-Computern zusammengesetzter Rechner.

N

Nesting
Verschachtelung; z.B. verschachteln von Unterprogrammen.
NMI
Non Maskable Interrupt. Eine Unterbrechung, die nicht gesperrt werden kann.

O

Odd
bedeutet ungerade. Die Zahl 3 ist zum Beispiel ungerade.

offener Kollektor
Schaltung, dessen Endtransistor einen herausgeführten, unbeschalteten Kollektor hat.
Oktal
Zahlendarstellung zur Basis 8
Output
Ausgabe

P

Packen
Dabei werden z.B. zwei Dezimalzahlen in einem Byte untergebracht
Parity
Parität, Gleichheit
Pascal
Eine höhere Programmiersprache, die für Lehrzwecke entworfen wurde und zunehmend Verbreitung findet. Siehe Kapitel Pascal/S
Pass
Lauf oder auch Durchgang, z.B. bei einem Übersetzungsvorgang
PE-Verfahren
Phase encoding. Verfahren bei der Aufzeichnung auf Datenträger, siehe Kapitel CAS
Pegel
Spannungsbereich
Peripherie-Geräte
Einheiten, die mit der Außenwelt eines Computer in Verbindung treten können.
Pipelining
Fließbandverarbeitung. An mehreren Stellen wird in kleinen Schritten gleichzeitig eine Verarbeitung vorgenommen. Dadurch kann man in Mikroprozessoren die Befehlszykluszeiten verkleinern.
PL/1
Programming Language 1. Eine höhere Programmiersprache, die zur Pascal-Familie gehört. Beispiel:

```
TEST: PROCEDURE OPTIONS(MAIN);
 DECLARE (A,B) FIXED DECIMAL
  (6,2),
  (COUNT) FIXED;
 A = 12.34; B = A + 1.02;
 PUT SKIP(2);
 DO COUNT=1 TO 10;
   PUT EDIT('I=',I,'B=',B) (F(5),F(6,2));
   B = B + 1.0;
 END;
END TEST;
```

PL/M
Ähnlich der Programmiersprache PL/1, jedoch eine Teilmenge daraus. Wurde vorwiegend für die 8080-Prozessorfamilie verwendet.
Plotter
Ein Gerät, ähnlich zu einem XY-Schreiber, für die Ausgabe von graphischen Darstellungen.
Pointer
Zeiger. Ein Speicherplatz, der eine Adresse eines anderen Speicherplatzes enthält.
Polling
Aufrufbetrieb. Darunter versteht man die ständige Abfrage, z.B. eines Peripheriegerätes, um so festzustellen, ob es schon fertig ist.
Port
Tor. Man meint damit Ein- oder Ausgabebausteine
Prellen
Mechanische Schalter berühren die Kontaktflächen beim Schließen oder Öffnen mehrere Male.
Programm
Ist eine Folge von Anweisungen (Befehlen), die zur Lösung eines Problems dienen sollen.
Programmiersprache
Eine Sprache zur Formulierung von Programmen, die automatisch (von einem Übersetzungsprogramm oder Interpreter) in Maschinensprache umgesetzt werden können.
Programmspeicher
Dort ist das auszuführende Programm abgelegt.
Programmzähler
Er legt die Adresse der Speicherzelle des nächsten Befehls fest.
PROM
Programmable Read Only Memory. Ein Festwertspeicher, der durch Anlegen elektrischer Impulse beschrieben werden kann.
Pseudobefehl
Eine Instruktion, die nicht im Befehlssatz des Prozessors vorhanden ist, und zur Steuerung des Assemblers dient.
Pull-Up-Widerstand
Ein Widerstand, nach +5V geschaltet, um z.B. eine offene Kollektorschaltung zu beschalten.

Q

Queue
Warteschlange. Daten werden in einer Warteschlage angesammelt, wenn sie noch nicht verarbeitet sind.

R

RAM
Random Access Memory. Speicher mit wahlfreiem Zugriff
Real time clock
Echtzeituhr
Redundanz
Teil einer Nachricht, die zur eigentlichen Information nichts mehr beiträgt.
refresh
Wiederauffrischen

Relokalisierbar
Ein Programm, das in verschiedenen Speicherbereichen lauffähig ist.
Reset
Rücksetzen
ROM
Read Only Memory; ein Speicher, den man nur auslesen kann.

S

scan
Abtasten
scrollen
Der Bildschirminhalt wird nach oben oder unten verschoben.
sedezimal
Zahlendarstellung zur Basis 16
select
Auswählen
sense
Abtasten
Simulator
Ein Programm, das einen Vorgang künstlich nachbildet.
Software
Hierunter versteht man alle Arten von Programmen, wie auch Texte und Informationen.
Source
Quelle
Space
Bei der seriellen Übertragung ist damit der logische Wert 0 gemeint.
Space-Taste
Leertaste auf der Tastatur, die einen Freiraum erzeugt.
Speicherbaustein
Ein Baustein, der Informationen behalten kann.
Stack
Stapelspeicher, Kellerspeicher. Siehe LIFO
State
Zustand
Statement
Anweisung, Befehl
statische RAMs
Speicher, die z.B. mit zwei Transistorzellen aufgebaut sind und wie ein bistabiles Flip-Flop arbeiten.
Steuerwerk
Dieser Teil des Computers kontrolliert die Ausführung sämtlicher Befehle, er wird auch mit Leitwerk bezeichnet.
Strukturierte Programmierung
Verfahrensweise, um einfach zu testende und verständliche Programme zu erzeugen.
Subroutine
Unterprogramm
Supervisor
Ein Organisationsprogramm

T

Tantal-Kondensator
Wird in Mikroprozessorschaltungen gerne zur Unterdrückung von Spannungsspitzen auf der Versorgungsleitung verwendet.
Terminal
Datenendstation
Text-Editor
Siehe Editor
Time sharing
Zeitscheibenverfahren. Dabei können mehrere Benutzer auf ein und denselben Computer zugreifen.
Timing-Diagramm
Zeitlicher Ablauf, bildlich dargestellt.
Trace
Ablaufverfolgung; Methode zur Fehlersuche in Programmen.
transfer
Übertragen
Tristate-Treiber
Ein Schaltkreis, der drei Zustände besitzt. Pegel auf 0, Pegel auf 1 oder offen (Pegel undefiniert).

U

UART
Universal Asynchronous Receiver/ Transmitter.
Die Schaltung dient der seriellen Übertragung.
Unit
Einheit, Gerät
Unterprogramm
Gleiche Befehlsfolgen, die in einem Programm mehrfach vorkommen, kann man zu Unterprogrammen zusammenfassen, und muß sie daher nur einmal abspeichern.

V

V24
Schnittstellen-Norm für serielle Signale.
Valid
gültig
Vektor Interrupt
Das auslösende Gerät gibt zusätzlich zur Interrupt-Anforderung auch noch eine Zieladresse vor.

W
Wait
Warten
Wired-Or
Eine logische Verknüpfung, die nur durch die Verdrahtung entsteht.
Worst case
Ungünstigster Fall
Wort
Zusammenfassung mehrerer Bits zu einer logischen Einheit.

Z
Z80-CPU
Ein Mikroprozessor-Baustein
Zeichengenerator
Der Zeichensatz für die Schriftdarstellung auf dem Bildschirm ist darin gespeichert.
Zugriff
Zugang z.B. in eine bestimmte Speicherzelle
Zyklus
Eine Anzahl von Schritten, die wiederholt werden und im Ablauf gewisse Ähnlichkeiten aufweisen

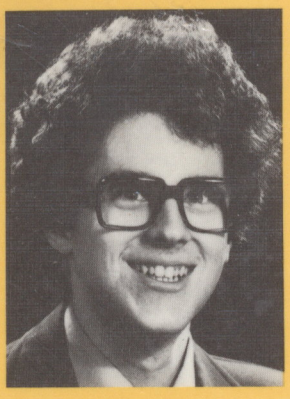
K Materialliste für Kapitel 3 bis 6

Folgende Systemausstattung ist für das Ausführen der Beispiele notwendig:

Kapitel	3.1	3.2	3.3	3.4	3.5	3.6	4	5	5.8	6.2/3	6.4
Ram	8K	8K	16K	16K	16K	16K	48K	16K	16K	16K	144K
Grundprog	ja	ja	ja	ja	ja	ja	ja	ja	ja	ja	ja
Pascal/S	-	-	-	-	-	-	ja	-	-	-	-
Gosi	-	-	-	-	-	-	-	ja	ja	-	-
Uform68K	-	-	-	-	-	-	-	-	-	ja	ja
CP/M68k	-	-	-	-	-	-	-	-	-	-	ja
BankBoot	-	-	-	-	-	-	-	-	-	-	ja
IOE	-	1x	1x	1x	1x	-	-	-	-	-	-
FLO2+Floppy	-	-	-	-	-	-	-	-	-	ja	ja
Joystick	-	-	-	ggf.	-	-	-	-	-	-	-
Tastenfeld	-	ja	-	ja	ja	-	-	-	-	-	-
Einzelteile	-	ja	ja	ja	ja	-	-	-	-	-	-
AD-DA	-	-	-	-	-	-	-	-	ja	-	-

L Sachwortverzeichnis

Klein
Die Prozessoren 68000 und 68008



Rolf-Dieter Klein

Nach der Modernisierung und Erweiterung des NDR-KLEIN-Computers wartet die Fachwelt auf dieses Werk.
Rolf-Dieter Klein führt in seiner bewährten Art in die Hardware der hochmodernen Prozessorfamilie 68000/68008 sowie der dazugehörigen Peripherietechnik ebenso ein, wie in die leistungsfähige Software. Der Leser lernt anfangs den Umgang mit dem Texteditor sowie dem Assembler und gelangt dann zu den höheren Programmiersprachen Pascal und Gosi. Die abschließende Erweiterung des Systems durch Floppy-Disk-Laufwerke führt hin zum Einsatz des professionellen Betriebssystems CP/M-68K.
Immer steht das Lernen durch Versuche im Vordergrund. Der modulare Computer eignet sich dafür in besonderer Weise und stellt außerdem sicher, nicht zu veralten.
Es ist die bewährte Art des Autors, die Mikrocomputertechnik im Selbstbau zu vermitteln. Auf diesem sicheren Weg erlangt der Anwender ein Niveau, von dem aus er erfolgreich in die professionelle Computertechnik einsteigen kann.

ISBN 3-7723-7651-7

Franzis'