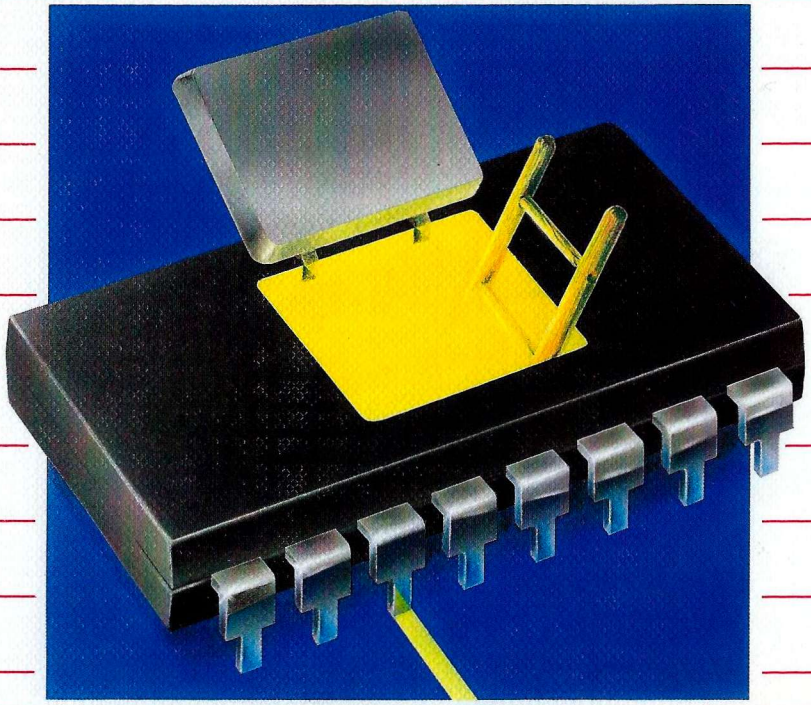
Helmut Ostermann

Maschinen- und Assemblersprache des M68000

Eine Einführung mit vielen Beispielen



CHIP WISSEN

Der Mikroprozessor 68000 von Motorola hat durch Computer wie Amiga, Macintosh, Atari ST und Sinclair QL in kurzer Zeit eine weite Verbreitung gefunden. Seine erstaunliche Leistungsfähigkeit und die klare Struktur der Adressierungsarten machen das Programmieren dieses Prozessors zur Freude.

Das Buch wendet sich an Leser mit Vorkenntnissen in der Programmierung eines Mikroprozessors, läßt aber den Anfänger nicht im Stich; er findet mit Hilfe des Glossars und eines Anhangs den Zugang.

Die zahlreichen Beispiele wurden auf Amiga, Sinclair QL und NDR-Klein-Computer entwickelt. Damit werden bewußt drei verschiedene Klassen von Computern und Betriebssystemen angesprochen.

VOGEL-BUCHVERLAG
WÜRZBURG

ISBN 3-8023-0174-9

Ein Buch von CHIP, dem Mikrocomputer-Magazin

CHIPWISSEN

Helmut Ostermann

Maschinen- und Assemblersprache des M68000

Eine Einführung mit vielen Beispielen für Amiga, NDR-Klein-Computer und Sinclair QL

VOGEL-BUCHVERLAG
WÜRZBURG

HELMUT OSTERMANN
1921 geboren. 1948 bis 1951 Lehrerausbildung an der PH Berlin, 1960 bis 1962 Weiterbildung zum Realschullehrer (Mathematik, Physik, Chemie) an der PH Oldenburg. Seit 1971 aktiver Umgang mit Computern, der sich auch in der Lehrtätigkeit auswirkte.

CIP-Kurztitelaufnahme der Deutschen Bibliothek

Ostermann, Helmut:
Maschinen- und Assemblersprache des M68000: e. Einf. mit vielen Beispielen für Amiga, NDR-Klein-Computer u. Sinclair QL / Helmut Ostermann. – 1. Aufl. –
Würzburg: Vogel, 1987.
(Chip-Wissen)
ISBN 3-8023-0174-9

ISBN 3-8023-0174-9
1. Auflage. 1987
Alle Rechte, auch der Übersetzung, vorbehalten. Kein Teil des Werkes darf in irgendeiner Form (Druck, Fotokopie, Mikrofilm oder einem anderen Verfahren) ohne schriftliche Genehmigung des Verlages reproduziert oder unter Verwendung elektronischer Systeme verarbeitet, vervielfältigt oder verbreitet werden. Hiervon sind die in §§ 53, 54 UrhG ausdrücklich genannten Ausnahmefälle nicht berührt.
Printed in Germany
Copyright 1987 by Vogel-Buchverlag Würzburg
Umschlaggrafik: Michael M. Kappenstein, Frankfurt/M.
Herstellung: Alois Erdl KG, Trostberg

Vorwort

Die Prozessoren der M68000-Familie sind noch relativ jung. Sie fanden Eingang in professionelle Anwendungen, in den Personal- und seit etwa einem Jahr auch in den Homecomputerbereich. Sie sind sehr leistungsstark und können auch Mehrbenutzersysteme verwalten. Der umfangreiche Befehlssatz ist dank des straffen Konzepts gut überschaubar, wodurch diese Prozessoren kaum schwieriger zu programmieren sind als andere.

Das Buch führt in die Programmierung dieser Prozessoren mit der Maschinen- und der Assemblersprache ein. Dabei wird der Sicherung der einzelnen Lernschritte durch konkrete Beispiele und auch durch Aufgaben und Lösungen der Vorrang vor einer nur formalen Systematik gegeben.

Nachvollziehbare Beispiele kann man nur mit einem Computer erarbeiten. Da weder für jeden Rechner mit CPU 68000 eine Einführung in die Maschinen-/Assemblersprache herausgebracht werden soll noch alle gängigen Maschinen in ein und demselben Buch vorgestellt werden können, wurden drei Rechner ausgewählt, die akzeptabel dokumentiert sind, den Sinclair QL, den NDR-Klein-Computer (beide mit CPU 68008) und den Commodore Amiga. Die Betriebssysteme dieser Computer sind so unterschiedlich strukturiert, daß sie exemplarisch drei verschiedene Kategorien vertreten.

Damit wird es möglich, das jeweils Besprochene am dafür am besten geeigneten Gerät vorzustellen, wobei in der Regel aber auch Varianten für die anderen Maschinen aufgezeigt werden sollen.

Es darf angenommen werden, daß die Mehrzahl der Leser bereits mit den Grundbegriffen der Maschinen-/Assemblerprogrammierung vertraut ist. Um ihnen ein zügiges Vorankommen zu ermöglichen, werden die Grundkenntnisse im laufenden Text vorausgesetzt. Aber auch die „Einsteiger" sollen nicht zu kurz kommen; ihnen vor allem sind Anhang B und das Glossar (Anhang D) zugedacht. So soll ihnen geholfen werden, die Anfangsschwierigkeiten und – wo vorhanden – auch die Schwellenangst zu überwinden.

Bei Abschluß des Manuskripts sei Herrn Steffen Garrelts herzlich gedankt. Er übernahm nicht nur die Korrektur, sondern hat auch wertvolle Anregungen zur Verbesserung der Feinstruktur gegeben.

Oldenburg Helmut Ostermann

Inhaltsverzeichnis

Vorwort 5

Einleitung 13
Zur Rechnerauswahl 13
Vorkenntnisse, Handbücher und Glossar 14
Ein wichtiges Hilfsprogramm 15

1 Die 68000-Familie 19
1.1 Der M68000 19
1.2 Die Datenformate 20
1.3 Der Adreßbus 20
1.4 Andere Mitglieder der 68000-Familie 20

2 Einführung in die Adressierungsarten 23
2.1 Effektive Adresse und Adressierungsarten 23
2.2 Das erste Programm 26
2.3 Die Besonderheiten des NDR-Computers 27
2.4 Das erste Programm für den Amiga 27
2.5 Das erste Programm für den QL 28

3 Speicherplatzunabhängige Programmierung (I) 29
3.1 Die Berechnung der Adreßdistanz 29
3.1.1 Ein Beispiel für den NDR-Computer 30
3.1.2 Ein Beispiel für den Amiga 31
3.1.3 Ein Beispiel für den QL 32
3.2 Das Programm wird reloziert 32

4 Speicherplatzunabhängige Programmierung (II) 33
4.1 Mehrfache Ablage im Speicher mit Postinkrement 33
4.2 Mit Predekrement 34
4.3 Ein Programm kopiert sich selbst 35

5 Vom Mnemonik zum Opcode 37
5.1 Die Codierung des Befehls MOVE 38
5.2 Assemblieren und Disassemblieren (I) 39

5.3 Die Codierung des Befehls MOVEQ 39
5.4 Die Codierung der Adressierungsart d(An,Rx.X) 40
5.5 Ein Experimentierprogramm zu d(An,Rx.X) 41
5.6 Zur Quelle-Ziel-Terminologie 42

6 Zählerkontrollierte Programmschleifen 43
6.1 DBRA-Schleifen 44
6.2 Schleifen mit Zweitbedingung (DBcc) 45

7 Vom Condition Code Register (CCR) 47
7.1 Die Flags des CCR (I) 47
7.2 Beobachtung des CCR 48
7.3 Flagkombinationen und Bedingungen 49
7.4 Die sechzehn Bedingungsschlüssel 50
7.5 Die Codierung des Befehls DBcc Dn,Label 51

8 Von bedingten und unbedingten Sprüngen 52
8.1 Die Codierung von BRA 52
8.2 Die Codierung von Bcc 53
8.3 Sortieren nach Merkmalen 54

9 Die Bedienung des Rolf-D.-Klein-Assemblers 55
9.1 Der Editor 55
9.2 Die ORG-Direktive 56
9.3 Die eigentliche Assemblierung 56
9.4 Der Programmablauf 57

10 Subroutinen und Stapel 58
10.1 Ein Beispiel 58
10.2 Stapel und Stapelzeiger 58
10.3 Ein Subroutinennest 60

11 Ein- und Ausgabe von Zeichen mit dem NDR-Computer 62
11.1 Die Subroutinen @CI und @CO2 62
11.2 Die Ausgabe von Strings 64

12 Stringausgabe mit dem QL 66
12.1 Die Grobstruktur des Programmbeispiels 66
12.2 Vektorisierte Routinen (I) 68
12.3 Die Ermittlung der QDOS-Version 69
12.4 Auswege für den Amiga-User 70

13 Der Metacomco-Assembler (QL) 71
13.1 Eine Direktive und zwei Eigenarten 71
13.2 Die Bedienung 72
13.3 Formales zum Metacomco-Assemblerprotokoll 74

14 Umwandlung von Hexadezimalzahlen in Strings 75
14.1 Problemlösung in einzelnen Schritten 75
14.2 Fertige Problemlösung mit dem NDR-Computer 77
14.3 Fertige Problemlösung mit dem QL 80
14.4 Das Subroutinennest ITOHn (QL) 82

15 Rotieren und Shiften 83
15.1 Die Shift-Befehle 84
15.2 Die Rotationsbefehle 84
15.3 Gemeinsamkeiten 84

16 Die logischen Verknüpfungen 86
16.1 AND 86
16.2 OR 86
16.3 EOR 87
16.4 NOT, NEG und NBCD 87

17 Die MOVEM-Befehle 88
17.1 Die Registerliste 88
17.2 Die Anordnung der Registerinhalte im Speicher 89
17.3 Die Codierung der Registerliste 89

18 Umwandlung von einem Zahlensystem ins andere 90
18.1 Umwandlung DEZ → HEX 90
18.2 Umwandlung HEX → DEZ 93
18.2.1 Hauptprogramm und Eingabe 93
18.2.2 Die eigentliche Umwandlung 95
18.2.3 Absuchen einer Tabelle und Anfügen 95
18.2.4 Stringaufbereiten und Ausgabe 97

19 Das Sieb des Eratosthenes 99
19.1 Problemanalyse 99
19.2 Das Eratosthenes-Programm für den NDR-Computer 101
19.3 Das Eratosthenes-Programm für den QL 104
19.4 Das Eratosthenes-Programm für den Amiga 107

20 Sortieren 110
20.1 Schriftzeichen sortieren 110
20.2 Die NDR-Routine @READ 112
20.3 Sortieren von Wörtern gleicher Maximallänge 115
20.4 Sortieren von Wörtern individueller Länge 119

21 Großes Sortierprogramm 123
21.1 Das Menü 123
21.2 Auswahl eines Menüpunkts 125
21.3 Alles oder einzelne Records löschen 126

21.4 Das eigentliche Entfernen eines Worts 128
21.5 Eingabe und Einfügen neuer Wörter 130
21.6 Die Aufzeichnung auf Diskette 132
21.7 Die übrigen Optionen 134

22 Fließkommazahlen (I) 136
22.1 Fließkommazahlen beim QL 136
22.2 Fließkommazahlen beim Amiga 138
22.3 Unsere Fließkomma-Norm 141
22.4 Vom Zahlenstapelkonzept 142
22.5 Die Konvertierung DEZ → FP 142
22.6 Die Konvertierung FP → DEZ 146
22.7 Programmablauf, Genauigkeit, Verbesserungsmöglichkeiten 150
22.8 Die Fließkomma-Addition 151
22.9 Die Additionsbefehle 153

23 Die Flags des CCR (II) 155
23.1 Die Flags und die Addition 155
23.2 Die Flags und die Subtraktion 158
23.3 Die Flags und die Bedingungsschlüssel 159

24 Subtraktion und Fließkomma-Subtraktion 162
24.1 Die Subtraktionsbefehle 162
24.2 Problemanalyse 163
24.3 Der Programmtorso 165

25 Vom K-Seka-Assembler (Amiga) 167
25.1 Inbetriebnahme des K-Seka-Assemblers 167
25.2 Vom Seka-Editor 168
25.3 Das Assemblieren 170
25.4 Der Programmablauf 170

26 Multiplikation und Fließkomma-Multiplikation 175
26.1 Die Multiplikationsbefehle 175
26.2 Mehrstellige Multiplikation 176
26.3 Die Fließkomma-Multiplikation (Amiga) 178

27 Division und Fließkomma-Division 181
27.1 Die beiden Divisionsbefehle 181
27.2 Noch ein Primzahlprogramm 182
27.3 Bitweise Division 184
27.4 Die Vervollständigung des FP-Rechenwerks 188

28 Die Türme von Hanoi 189
28.1 Problemanalyse 189
28.2 Zur grafischen Darstellung auf dem Bildschirm 193

28.3 Das Variablen-Handling 194
28.4 Die vorbereitenden Routinen 195
28.5 Angewandte Grafik und Analyse 198
28.6 Der Scheibentransport 200
28.7 Vom Status des Programms 206
28.8 Manöverkritik 207
28.9 Die Türme von Hanoi auf dem NDR-Computer 208

29 Disassembler 214
29.1 Entwurf eines Disassemblers 215
29.2 Speicher- und Registerdisposition 218
29.3 Die Subroutine OPCODE (20AB0) 218
29.4 Die Subroutine SELECT (20AEA) 220
29.5 Die Subroutine CONBWL (20E2E) 222
29.6 Die Subroutine AUSGEB (20BAC) 224
29.7 Die Befehle der Gruppe 2 (200DC) 224
29.8 Die TRAP-Befehle (GRUP3: 20124) 238
29.9 Die Befehle der Gruppe 4 (20148) 238
29.10 Die Superroutine EAXXEX (20C80) 240
29.11 Weitere Routinen 240
29.12 Die Decodierung der Registerliste (20EAE) 242
29.13 Die Übertragung auf andere Computer 248
29.13.1 Realisierung in der Assemblersprache 248
29.13.2 Übertragung in andere Sprachen 248

30 Nachlese der nicht privilegierten Befehle 254
30.1 Bit/Byte-Abfrage und Bit/Byte-Manipulation 254
30.2 Weiteres zum Stapel 254
30.3 Datentransfer von der und zur Peripherie (MOVEP) 258

31 Ausnahmen vom normalen Programmablauf (I) 259
31.1 Das Statusregister 259
31.2 Die Interrupt-Ebenen 260
31.3 Die Interrupt-Ebenen des QL 263

32 Ausnahmen vom normalen Programmablauf (II): Die TRAP-Befehle . . . 265
32.1 User- und Supervisor-Modus 265
32.2 Aufgabe und Organisation der TRAPs 266
32.3 Vektorisierte Routinen (II) 268
32.4 Ablauf eines TRAP-Befehls und Rückkehr 270

33 Experiment im Supervisor-Modus 271
33.1 Von den privilegierten Befehlen 271
33.2 TRAP #0 beim QL 272
33.3 Wo befindet sich der Supervisor-Stapel? 272
33.4 Modus-Experiment mit dem NDR-Computer 274

34 Die Anmeldung eines Programms beim Betriebssystem QDOS 275
34.1 Projekt «Bremer Stadtmusikanten» 275
34.2 Der Programmkopf und die Priorität 275
34.3 Priorität und Scheduler (I) 277
34.4 Die übrigen Programmteile 277

35 Multitasking 279
35.1 Die drei anderen Tiere 279
35.2 Der Job-Kontrollblock 280
35.3 Priorität und Scheduler (II) 282

36 Tracing 284
36.1 Zum Programmentwurf 284
36.2 Vorbereitung und Einschalten des Trace-Modus 286
36.3 Von der Console ins Register 288
36.4 Wurzelziehen durch Bit-KO 288
36.5 Die Anzeige im Trace-Modus 290
36.6 Die Selbstentfernung des Programms 290

37 Another self-cloning program 293
37.1 Programmentwurf 293
37.2 Die Tochterprogramme 295
37.3 Das Masterprogramm 297
37.4 Die Jobproduzierroutine CREAJOB 297
37.5 Der Gesamtablauf 301

38 The self-cloning program von Adrian Dickens 303

39 Die unvollendete Raummühle (für den NDR-Computer) 304
39.1 Die Darstellung auf dem Bildschirm 304
39.2 Interne Codierung 306
39.3 Zwickmühlen 309
39.4 Die Strategie 309
39.5 Die Siegmeldung 310
39.6 Weiterer Ausbau 311

40 Zum Ausklang 328

Anhang 329
A) Literaturverzeichnis 329
B) Programmierung eines Mikroprozessors 331
C) Lösungen und Lösungsvorschläge zu den Aufgaben 333
D) Glossar 353
E) Modus-Experiment mit dem NDR-Computer 357

Stichwortverzeichnis 359

Einleitung

Man erlernt eine Fremdsprache nicht, indem man sich zuerst alle Substantive und ihre Deklination, dann alle Verben und ihre Konjugation und schließlich Aussprache, Satzbau und typische Redensarten einprägt. Sinnvollerweise wird man von all diesem zunächst nur je ein bißchen aufnehmen und damit einfache Sätze bilden. Der Lernzuwachs wird anfangs nicht von der Systematik bestimmt, sondern von praktischen Bedürfnissen und der größtmöglichen Erweiterung der Ausdrucksfähigkeit.

Allmählich muß dann aber auch die Systematik einfließen, damit man die verbleibenden Feinheiten in den Nachschlagewerken finden und nachlesen kann. Unser Lehrgang soll diesem bewährten Prinzip folgen.

Zur Rechnerauswahl

So wie man eine lebende Sprache am besten durch Hören und Sprechen erlernt, kann man eine Programmiersprache am besten mit und an einem Computer erlernen. Dementsprechend werden im folgenden bewußt viele Beispiele an konkreten Rechnern vorgestellt, wofür drei verschiedene und für ihre Gruppe typische Mikrocomputer ausgesucht wurden: Amiga, QL und der NDR-Klein-Computer. Abhängig von der Güte der verfügbaren Dokumentation wird sich das Besprochene ohne große Schwierigkeiten auf andere Maschinen übertragen lassen.

Die dem Verfasser bekannten Rechner mit dem M68000 lassen sich in vier Kategorien einteilen:

a) Maschinen oberhalb von 10000 DM;
b) Computer mit grafikorientierten Betriebssystemen («Benutzeroberfläche») sind für unser Vorhaben nicht besonders gut geeignet: Das Betriebssystem ist undurchsichtig, die Dokumentation auf der Prozessorebene oft

dürftig bis sehr schlecht – oder teuer und hypertroph (man suche die Nadel im Heuhaufen); zudem sind einfache Ein- und Ausgaben oft nur umständlich zu realisieren. Im Herbst 1985 wurde ein 512-KB-Rechner ausgeliefert, der unter BASIC nur noch 3 KB für den Benutzer übrigließ; Urvater PET stellte 1978 von seinen 8 KB RAM 7 zur Verfügung!

c) Sinclair QL: Eine preisgünstige Maschine mit akzeptabel dokumentiertem 68000-spezifischen Betriebssystem. Am Beispiel des QL soll im zweiten Teil auf die Besonderheiten von Mehrbenutzersystemen eingegangen werden.

d) Selbstbaucomputer. Der Verfasser behauptet nicht, daß ihm der Bau ohne alle Probleme gelang. Er lernte dann aber im NDR-Computer einen Rechner kennen, der auf Prozessorebene mit beachtlichem Komfort große Datenmengen verwalten kann; das Ergebnis kam der Idealvorstellung sehr nahe.

Sollte es beim Nachvollziehen der Beispiele Schwierigkeiten geben, darf sich der Leser darauf verlassen, daß alle Programme ausprobiert wurden und wie beschrieben abliefen. Die QL-Programme wurden mit der englischen Version (1.03) entwickelt.

Vorkenntnisse, Handbücher und Glossar

Obgleich der M68000 dank seines Befehlsreichtums letztlich nicht schwieriger zu programmieren ist als andere Prozessoren, wird angenommen, daß die meisten Leser bereits Vorkenntnisse in der Maschinenprogrammierung mitbringen. Dementsprechend geht es im ersten Kapitel gleich richtig los. Der Newcomer wird besonders auf die Anhänge B und D verwiesen. Er findet außerdem im Literaturverzeichnis Hinweise auf Grundlagenbücher.

Da es nicht möglich ist, in einem dreihundertseitigen Buch den Inhalt eines 576seitigen Handbuchs, eine Einführung und auch noch Beispiele zu bieten, kommt der an mehr als einem summarischen Überblick interessierte Leser an der Anschaffung eines Handbuchs nicht vorbei – eine Investition von bleibendem Wert!

Ein wichtiges Hilfsprogramm

Die bei der Rechnerauswahl unter b) erwähnten Schwierigkeiten lassen sich im Anfang leicht und bequem mit einem Monitorprogramm (siehe Listings P.0.1 und P.0.2) umgehen. Drei Befehle sind vorgesehen:

C-Befehl Beispiel: C37F00
Dieser Befehl verändert (change) den Speicherinhalt und dient zur Eingabe des Maschinenprogramms (Objektcode) und der benötigten Daten. Im Beispiel erscheint auf dem Bildschirm die Aufforderung 037F00: mit blinkendem Cursor. Gibt man nun z.B. 7000 (und cr) ein, wird in den Speicherstellen mit den Adressen 037F00 und 037F01 das Datenwort 7000 abgelegt. Die Aufforderung 037F02: beantwortet man z.B. mit 4E75. So geht es endlos weiter, bis man beim QL «/» eingibt oder beim Amiga einfach cr drückt.

G-Befehl Beispiel: G37F00
Dieser Befehl startet das bereitgestellte Programm bei der angegebenen Adresse (G steht für «go»). Liegt kein Fehler vor, kehrt der Ablauf zur Kommandoebene des Monitorprogramms zurück.

H-Befehl Beispiel: H37F00
Hiermit wird ein Speicherauszug (Hexdump) von der spezifizierten Adresse an ausgegeben. Die sonst endlose Ausgabe wird durch cr beendet. Diese Schleife ist auch gut geeignet, um sich ein wenig im Innenleben seines Computers umzusehen.

Probelauf

Amiga-Besitzer sollten lediglich 4E75 eingeben. Dieser Maschinenbefehl leitet den Ablauf aus dem Maschinenprogramm nach BASIC, also ins Monitorprogramm, zurück.

QL-Besitzer müssen zuerst 7000 und dann 4E75 eingeben, um den gleichen Ablauf zu erzielen.

NDR-Besitzer brauchen kein derartiges Hilfsprogramm. Sie wählen dort aus dem angebotenen Menü die Option 1.1 «aendern», wählen die Standardadresse $9C00 und geben dort $4E75.W oder in zwei Teilen $4E und $75 ein, gehen mit M zurück zum Menü, von wo aus man mit Option 1.2 «starten» zur Starthilfe gelangt und als Startadresse $9C00 eingibt. Der

Bildschirm wird dunkel; nur rechts unten der Hinweis, daß man mit M wieder ins Menü gelangt. Option 1.3 «ansehen» und Eingabe der Von-Adresse liefern einen Speicherauszug, der mit 4E75 beginnt, gefolgt von irgendwelchen zufälligen Daten. Hätten wir mehr als die bloße Rückkehr zur aufrufenden Ebene vorgeschrieben, könnten wir auf diesem Wege auch erkennen, was unser Programm anrichtete.

Zu den beiden Monitorprogrammen

Auf die erfreuliche Strukturierbarkeit des Amiga-BASIC braucht nicht eingegangen zu werden; sie ist evident. Da das Hexadezimalformat &H4E75 nur für Zahlen < $10000 ausgelegt ist, werden die mehr als vierstelligen Adressen als Strings eingelesen, deren Zahlenwert durch die Subroutine hxdz ermittelt wird.

Wesentlicher ist, daß man bei typischen 68000er Betriebssystemen nie recht weiß, wo denn eigentlich die eingegebenen Programme stehen (das ist auch gar nicht nötig!). Es wäre gefährlich, sie irgendwo im Speicher abzulegen; ein Programm- oder gar Systemabsturz wäre dann sehr wahrscheinlich. Hier wird nun der String code$ wie jede andere vom Programm benutzte Variable vom Betriebssystem kontrolliert im Speicher untergebracht; er ist als 768mal 00 definiert, also nicht auf dem Bildschirm darstellbar. Seine Anfangsadresse wird von der SADD-Funktion ermittelt und der Variablen anfad übergeben. Nun wird dem User angezeigt, in welchem Speicherbereich er seine Maschinenprogramme unterbringen kann, nämlich in diesem String, der natürlich nicht anderweitig verarbeitet werden darf. Das Maschinenprogramm wird schließlich mit CALL aufgerufen. Für nicht allzu lange Programme ist das ein durchaus akzeptabler (und ausbaufähiger – siehe Handbuch) Weg, der zudem die spröde Ein- und Ausgabeproblematik umgeht.

Das Monitorprogramm für den QL

Abgesehen von den Eigenarten des SuperBASIC des QL ist nur noch der Befehl RESPR zu erwähnen; er reserviert Speicherplatz für eine Prozedur, hier 2048 Byte. Als Funktion übergibt er die Anfangsadresse wie eben ausgeführt.

```
REM monitor fuer Amiga
        code$=STRING$(768,0): anfad=SADD(code$)
jobdef: PRINT "Monitorbefehle Ccr G Hcr"
        PRINT " frei von  "; HEX$(anfad);
        PRINT        " bis "; HEX$(anfad+500): PRINT
INPUT"Befehl: ";bef$: l=LEN(bef$)
        b$=LEFT$(bef$,1): h$=RIGHT$(bef$,l-1)
        GOSUB hxdz: ad=dz
          IF b$="C" THEN change
          IF b$="G" THEN laufen
          IF b$="H" THEN hxdump
        GOTO jobdef

change: PRINT  HEX$(ad);":";: INPUT"";h$
        IF h$="" THEN GOTO jobdef
        GOSUB hxdz: POKEW ad,dz
        ad=ad+2: GOTO change

laufen: CALL ad: GOTO jobdef

hxdump: GOSUB zeile: IF INKEY$<>"" THEN GOTO jobdef
        GOTO hxdump

hxdz:   dz=0: l=LEN(h$)
        FOR li=1 TO l: hz=ASC(MID$(h$,li,1))-48
          dz=dz*16+hz+7*(hz>9)
        NEXT li: RETURN

zeile:  GOSUB adres: FOR ni=1 TO 8
                        GOSUB nxtwrt
        NEXT ni: PRINT  dz$; zz$: RETURN

adres:  dz$=RIGHT$("0000"+HEX$(ad)+" ",6)
        zz$="": RETURN

nxtwrt: dz$=dz$+RIGHT$("0000"+HEX$(PEEKW(ad))+" ",5)
        GOSUB nxtchr: GOSUB nxtchr: RETURN

nxtchr: pw=PEEK(ad): nz=46
        IF (pw>31)*(pw<127) THEN nz=pw
        zz$=zz$+CHR$(nz): ad=ad+1: RETURN
```

P.0.1 Das Monitorprogramm für den Amiga

```
100 CLS: PRINT"QL-Mini-Monitor-Befehle: C G H / cr)
110 anfad = RESPR(2048): C6X(anfad)
120                 PRINT "Frei von ";CX$
130 C6X(anfad+2047): PRINT " bis ";CX$
140 INPUT"*";bef$
150 b$ = bef$(1): ad = hexdez(bef$(2 TO LEN(bef$)))
160       IF b$="G" THEN CALL ad: GO TO 140
170       IF b$="C" THEN GO TO 210
180       IF b$="H" THEN GO TO 250
190 GO TO 140
200 REMark ---------------------------------- C-Befehl
210 C6X(ad): ad$=CX$: PRINT ad$;": ";: INPUT"";hw$
220       IF hw$="/" THEN GO TO 140
230       POKE_W ad,hexdez(hw$): ad=ad+2: GO TO 210
240 REMark ---------------------------------- H-Befehl
250 C6X(ad): dz$=CX$: dzz$=""
260 FOR s=1 TO 4
270   pw=PEEK_W(ad): pw=pw+65536*(pw<0)
280   C4X(pw): ad=ad+2: dz$=dz$&" "&CX$
290   dzz(ad): dzz(ad+1)
300 NEXT s
310 PRINT dz$;" ";dzz$
320 k$=INKEY$: IF k$="" THEN GO TO 250
330 GO TO 140
340 REMark ==========================================
350 DEFine FuNction hexdez(hex$)
360     dez=0: FOR j=1 TO LEN(hex$)
370     hexzif=CODE(hex$(j))-48
380     hexzif=hexzif-7*(hexzif>9)
390     dez=dez*16 + hexzif: NEXT j
400     RETurn dez
410 END DEFine
420 REMark ------------------------------------------
430 DEFine PROCedure C6X(rz): sz=6: GO SUB 490
440     END DEFine
450 DEFine PROCedure C4X(rz): sz=4: GO SUB 490
460     END DEFine
470 DEFine PROCedure C2X(rz): sz=2: GO SUB 490
480     END DEFine
490     CX$="": sw=16^(sz-1): FOR i=1 TO sz
500     q=INT(rz/sw): rz=rz-q*sw: sw=sw/16
510     CX$=CX$&CHR$(q+48+7*(q>9)): NEXT i
520     RETurn
530 REMark ------------------------------------------
540 DEFine PROCedure dzz(adr): pw=PEEK(adr-2)
550     IF (pw<32)+(pw>126) THEN pw=46
560     dzz$=dzz$&CHR$(pw)
570 END DEFine
```

P.0.2 Das Monitorprogramm für den QL

1
Die 68000-Familie

Motorola brachte den Prototyp XC68000 schon 1979 heraus; er faszinierte durch seine Leistungsfähigkeit. Ab 1983 gelangte der Standardtyp M68000 in die Reichweite von Hobbyisten. Der Name ist nach KOLLER (siehe Literaturverzeichnis) auf die 68000 Transistoren in seinem Chip zurückzuführen.

1.1 Der M68000

Zu den hervorragenden Features des M68000 gehören u.a.:

a) seine sehr potenten Befehle: Vieles, wozu man bei anderen Prozessoren zwei, drei oder sogar 30 Befehle benötigt, kann er mit einem einzigen bewältigen.
b) Seine Befehle sind sehr systematisch aufgebaut; die daraus resultierenden 57 oder 58 Befehlsgruppen sind gut zu überschauen.
c) Ihre effektive Vielfalt ergibt sich aus der sehr durchdachten Philosophie der «Adressierungsarten», deren es rund 12 gibt.
d) Viele Befehle haben eine Zwei-Adreß-Struktur: WAS WOHER WOHIN? (WAS soll geschehen? WOHER soll der Quelloperand genommen werden? WOHIN soll das Resultat abgelegt werden – Ziel?). Hieraus und aus den vorher genannten Zahlen ergibt sich eine grobe Abschätzung der Befehlsvielfalt: 58 * 12 * 12 = 8352.
e) Der M68000 kann auch Mehrbenutzersysteme verwalten; auf dem QL können bis zu 56 Programme (scheinbar) gleichzeitig ablaufen!
f) Der M68000 ist ein echter 16-bit-Prozessor; sein Datenbus ist 16 bit breit.

1.2 Die Datenformate

Dem 16 bit breiten Datenbus entsprechen 16 Bit-Daten; daher ist das Standarddatum des M68000 das zwei Bytes umfassende Datenwort; er kann mit einem einzigen Speicherzugriff also eines von 65536 verschiedenen Daten transferieren.

Die 16 Register dieses Prozessors sind sogar 32 bit breit; sie können also zwei Datenwörter nebeneinander bzw. gleichzeitig aufnehmen und i. a. auch verarbeiten. Ein 32 bit breites Datum heißt Langwort. Die Suffixe .B, .W oder .L zeigen an, ob der Prozessor den betreffenden Befehl im unteren Viertel, in der unteren Hälfte oder im vollen Umfang seiner Registerbreite ausführen soll.

Damit erhöht sich die Grobabschätzung des Befehlsvorrats auf 25056. Der Ansatz vereinfacht die tatsächliche Befehlsvielfalt, liefert aber wohl eine realistische Größenordnung.

1.3 Der Adreßbus

Der Adreßbus des M68000 ist effektiv 24 bit breit; er umfaßt die Adreßleitungen A1 ... A23. Die fehlende Leitung A0 wird durch zwei Steuerleitungen – $\overline{\text{LDS}}$ und $\overline{\text{UDS}}$ (lower/upper data strobe) – ersetzt, die anzeigen, ob das niederwertige oder das höherwertige Halbwort (Byte) angesprochen werden soll. Dieses Konzept macht deutlich, weshalb in aller Regel Datenwörter und -langwörter nicht aus oder zu ungeraden Adressen bewegt werden dürfen.

Da 24 bit 16777216 verschiedene Zahlenwerte darstellen können, kann der M68000 einen Speicher von 16 MB (Megabyte) adressieren.

1.4 Andere Mitglieder der 68000-Familie

Der M68008 ist eine etwas abgemagerte Version des M68000. Abgesehen von einer kleinen Abweichung in der Interruptstruktur wird er genauso programmiert wie der Standardtyp. Sein Datenbus ist nur 8 bit breit; um ein Datenwort aus dem Speicher zu holen, sind also zwei Lesezugriffe nötig. Das verringert seine Arbeitsgeschwindigkeit auf etwa 60 % des Grundtyps, vereinfacht aber den Bau von Mikrocomputern so sehr, daß sich schon für rund 500 DM sehr respektable Maschinen bauen lassen.

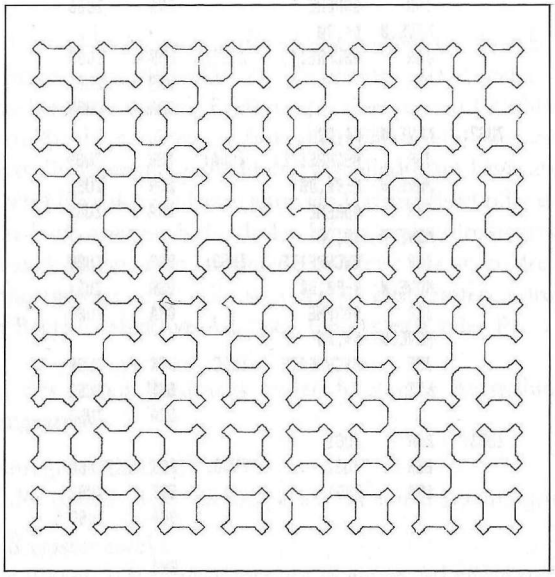
Auch der Adreßbus des M68008 ist schmaler. Mit seinen 20 Leitungen kann er 1 048 576 verschiedene Adressen ansprechen, also einen Speicher von 1 MB verwalten.

Der M68010 weist gegenüber dem Grundtyp noch drei zusätzliche Register auf. Dementsprechend bietet er noch einige Befehle mehr.

Der M68020 verfügt über eine nochmals kräftig gesteigerte Potenz: Sein Adreß- und sein Datenbus sind je 32 bit breit; er kann also sogar Langwörter in einem einzigen Zugriff transferieren. Außerdem hat er in seinem Innern noch einen verdeckten Speicher (cache), der Schleifen (ohne Speicherzugriff) beschleunigt.

Eine nochmalige Steigerung bietet der MC68030, über den Sie am besten in CHIP 2/87 auf Seite 48 f. nachlesen.

Es gibt auch noch einen M68012, doch können wir auf ihn und die anderen Sonderformen – außer auf den M68008 – nicht weiter eingehen.



Zugabe: Sierpinski-Kurve

```
* SIERPINSKI  03.02.87
INIT:   MOVE.W  #40,D1
        MOVE.W  #30,D2
        MOVE.W  #45,D3
        JSR     @SET
WABEN:  BSR     ZUGD
        BSR     ZUG2
        RTS
ZUG1:   MOVE.W  #4,D0
        JSR     @SCHREITE
        MOVE.W  #45,D0
        JSR     @DREHE
        MOVE.W  #16,D0
        JSR     @SCHREITE
        MOVE.W  #45,D0
        JSR     @DREHE
        MOVE.W  #4,D0
        JSR     @SCHREITE
        RTS
ZUG2:   MOVE.W  #4,D0
        JSR     @SCHREITE
        MOVE.W  #-90,D0
        JSR     @DREHE
        MOVE.W  #8,D0
        JSR     @SCHREITE
        MOVE.W  #-90,D0
        JSR     @DREHE
        MOVE.W  #4,D0
        JSR     @SCHREITE
        RTS
ZUG3:   BSR     ZUG1
        BSR     ZUG2
        BRA     ZUG1
ZUG4:   BSR     ZUG3
        BSR     ZUG2
        BRA     ZUG3
ZUG5:   BSR     ZUG3
        BSR     ZUG1
        BRA     ZUG3
ZUG6:   BSR     ZUG5
        BSR     ZUG2
        BRA     ZUG5
ZUG7:   BSR     ZUG6
        BSR     ZUG2
        BRA     ZUG6
ZUG8:   BSR     ZUG6
        BSR     ZUG1
        BRA     ZUG6
ZUG9:   BSR     ZUG8
        BSR     ZUG2
        BRA     ZUG8
ZUGA:   BSR     ZUG9
        BSR     ZUG2
        BRA     ZUG9
ZUGB:   BSR     ZUG9
        BSR     ZUG1
        BRA     ZUG9
ZUGC:   BSR     ZUGB
        BSR     ZUG2
        BRA     ZUGB
ZUGD:   BSR     ZUGC
        BSR     ZUG2
        BRA     ZUGC

        END
```

Das Sierpinski-Programm

2
Einführung in die Adressierungsarten

Dieses Kapitel verfolgt zwei Ziele: Der zentrale Begriff «Adressierungsart» soll – natürlich an einem konkreten Beispiel – mit erstem Inhalt versehen werden. Zum anderen soll an diesem Beispiel konkret gezeigt werden, wie man ein und dasselbe Programm auf den drei Beispielmaschinen ablaufen lassen kann.

2.1 Effektive Adresse und Adressierungsarten

Beim M68000 schließt der Begriff Adresse neben den Speicherstellen auch noch die internen Speicherzellen des Prozessors (die Register) ein. Bild 2.1 gibt – zunächst ohne weitere Erklärung – einen ersten Überblick.

Zu einem Befehl gehören in fast allen Fällen ein oder zwei Angaben darüber, wo die Operanden zu finden (Quelladresse) bzw. abzulegen sind (Zieladresse). Eine solche Adresse kann ein Datenregister oder ein Adreßregister sein; sie kann aber auch durch den Inhalt eines Adreßregisters bestimmt werden, ja sie kann sogar in verschiedener Weise aus bis zu drei Komponenten zusammengesetzt sein. Alle so oder so bestimmten Adressen werden unter dem Begriff «effektive Adresse» (abg. <ea> oder EA) zusammengefaßt.

Nur um des ersten Eindrucks willen folgt eine Aufstellung der zwölf Adressierungsarten:

Dn – Datenregister direkt
Hier steht der (Quell- oder Ziel-)Operand in einem Datenregister.

An – Adreßregister direkt
Hier ist der (Quell- oder Ziel-)Operand in einem Adreßregister zu finden.

(An) – Adreßregister indirekt
Hier steht die Adresse des Operanden in einem Adreßregister.

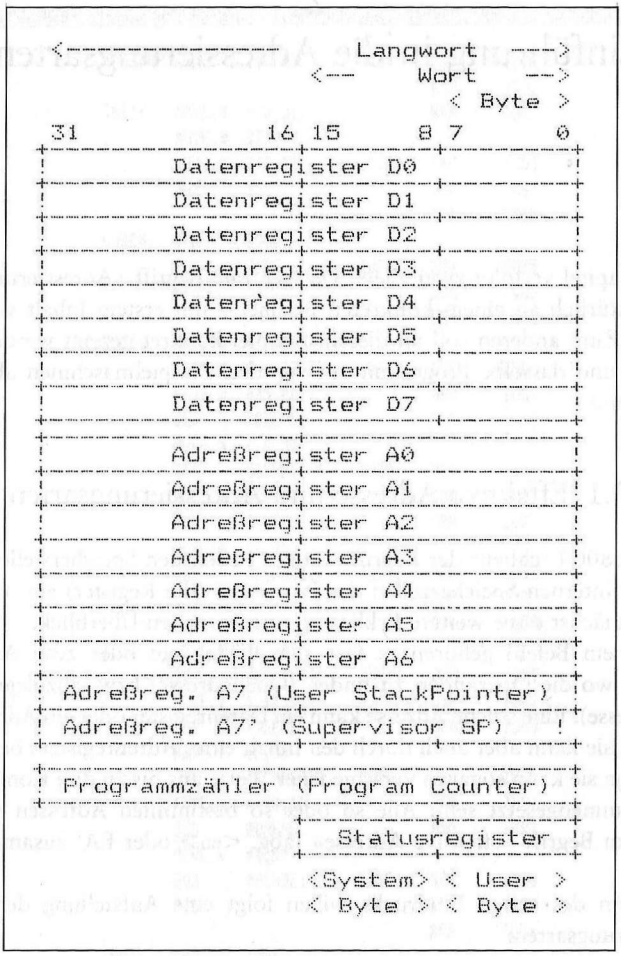


Bild 2.1 Die Register des M68000

(An)+ – Adreßregister indirekt mit Postinkrement
Zum Unterschied von (An) wird hier die in An stehende Adresse nach (daher «post») der Operation um die dem Datenformat entsprechende Anzahl von Bytes inkrementiert (erhöht).

–(An) – Adreßregister indirekt mit Predekrement
Zum Unterschied von (An)+ wird hier vor (daher «pre») der Operation die in An eingeschriebene Adresse um die dem Datenformat entsprechende Anzahl von Bytes dekrementiert (vermindert).

d(An) – Adreßregister indirekt mit Adreßdistanz
Hier errechnet sich die effektive Adresse aus der in An eingeschriebenen Zahl, vermehrt oder vermindert um die vorzeichenbehaftete 16-bit-Adreßdistanz:

$$0000 \leq d \leq 7FFF \text{ entspricht } 0 \leq d \leq 32767,$$
$$8000 \leq d \leq 0000 \text{ entspricht } -32768 \leq d \leq 0.$$

d(An,Rx.X) – Adreßregister indirekt mit Adreßdistanz und Index
Hier setzt sich die effektive Adresse aus der in An eingeschriebenen Zahl und dem in einem Daten- oder Adreßregister stehenden .W- oder .L-Datum und der vorzeichenbehafteten 8-bit-Distanz zusammen.

$xxxx – absolut kurz
Hier wird die effektive Adresse unmittelbar durch die 16-bit-Hausnummer der gemeinten Speicherzelle(n) bezeichnet. Das vorangestellte Dollarzeichen kennzeichnet nach Motorola-Norm eine Hexadezimalzahl. Der RDK-Assembler des NDR-Computers akzeptiert hier auch Dezimalzahlen – natürlich ohne «$» davor.

$xxxxxxxx – absolut lang
Hier darf die absolute Adresse ein Langwort sein.

d(PC) – Programmzählerrelativ mit Adreßdistanz
Die effektive Adresse ergibt sich als Summe vom momentanen Programmzählerstand und der vorzeichenbehafteten 16-bit-Distanz.

d(PC,Rx.X) – Programmzählerrelativ mit Adreßdistanz und Index
Hier setzt sich die effektive Adresse aus dem Programmzählerstand und dem in einem Daten- oder Adreßregister eingeschriebenen .W- oder .L-Datum und der vorzeichenbehafteten 8-bit-Adreßdistanz zusammen.

#*Konstante* – Konstantenadressierung
Hier ist der eine Operand (notwendigerweise der Quelloperand) unmittelbar als Zahl (Konstante) vorgegeben.

Außer diesen zwölf (sozusagen «regulären») Adressierungsarten besteht in einigen Sonderfällen auch die Möglichkeit, die Kontrollregister CCR und SR sowie den Supervisor-Stackpointer SSP (beides siehe später) anzusprechen.

2.2 Das erste Programm

P.2.1 besteht nur aus zwei Befehlen. Der erste weist einer .L-Speicherstelle (bzw. vier aufeinanderfolgenden .B-Speicherstellen), deren Adresse direkt durch ihre (erste) «Hausnummer», also absolut bestimmt ist, eine unmittelbar gegebene Konstante zu; Ziel ist hierbei die erste freie Speicherstelle hinter dem Programm. Dieser erste Befehl läßt sich verallgemeinern zu

MOVE.L #Konstante,$xxxxxxxx

Er benutzt die beiden Adressierungsarten «Konstantenadressierung» (auch «immediate») und «absolut lang». Der zweite Befehl leitet den Programmablauf zum Betriebssystem zurück.

(Leser, die nicht mit den Begriffen Opcode, Mnemonik usw. vertraut sind, werden auf die Anhänge B und D verwiesen.)

```
020000  23FC 2807 1986 0002 000C   MOVE.L  #$28071986,$2000C
02000A  4E75                       RTS
```

(Version a)

```
009C00  23FC 2807 1986 0000 9C0C   MOVE.L  #$28071986,$09C0C
009C0A  4E75                       RTS
```

(Version b)

```
009C00  21FC 2807 1986 9C0A        MOVE.L  #$28071986,$9C0A.S
009C08  4E75                       RTS
```

(Version c)

P.2.1 Das erste Programm für den NDR

2.3 Die Besonderheiten des NDR-Computers

Die Version a des ersten Programms wurde mit dem residenten Assembler (unter Systema-DOS) erstellt. Wir kommen darauf noch sehr genau zurück; die Erwähnung soll nur die Auswahl der Adressen begründen. Wir kümmern uns auch noch nicht weiter um den Weg vom Mnemonik zum Opcode, der schon bald besprochen wird, sondern tippen – wie in der Einleitung unter «Probelauf» beschrieben – den Objektcode ein, starten das Programm und fordern schließlich den Speicherauszug an, der so aussieht:

020000 23 FC 28 07 19 86 00 02 00 0C 4E 75 28 07 19 86

Man erkennt, daß in die adressierte .L-Speicherstelle tatsächlich das gewählte Datum eingetragen wurde.

Version b benutzt die Standardadresse für Systeme ohne Floppy. Ein Vergleich der Versionen a und b zeigt sofort den ersten Nachteil der Absolutadressierung: Wenn man ein und dasselbe Programm (und das liegt doch eigentlich hier vor!) an verschiedenen Stellen im Speicher bereitstellen will, müssen die absoluten Adressen an die neue «Location» angepaßt werden. Der M68000 bietet wesentlich potentere Adressierungsarten, die zudem noch Bytes sparen. Die absolute Adressierung wird man wohl nur anwenden, wenn man ein Programm in ein EPROM einbrennen will, für das ein unverrückbar fester Platz vorgesehen ist.

Version c entsprang der Versuchung, die Adressierungsart absolut kurz einzusetzen, weil in b die Zieladresse (hexadezimal) nur vierstellig (also 16-bit-Adresse) ist. Hier ist Vorsicht geboten. Das Programm leistet nicht das Erwartete, denn auch hier gilt eine Zahl (bzw. Adresse) als negativ, wenn ihr höchstwertiges Bit gesetzt ist; und das ist in 1001 1100 0000 1010 der Fall. Es ist also 009C0A ≠ 9C0A = FF9C0A (= −$63F6). Ganz in der Nähe dieses Bereichs sind aber die I/O-Ports angesiedelt!

2.4 Das erste Programm für den Amiga

Im konkreten Probelauf für diese Anleitung meldete das Monitorprogramm den Bereich 35B9A ... 35D8E als frei für Maschinencode. Glücklicherweise braucht das Programm nicht in der ersten freien Speicherstelle zu beginnen, so daß sich die Anpassungsarbeit begrenzen läßt, indem man als Startadresse 35C00 wählt:

```
035C00  23FC 2807 1986 0003 5C0C  MOVE.L  #$28071986,$35C0C
035C0A  4E75                      RTS
```

P.2.2 Das erste Programm für den Amiga

Nach Programmablauf gleicht der Speicherauszug dem eben Dargestellten; nur die absoluten Adressen stimmen nicht überein.

2.5 Das erste Programm für den QL

Hier sind drei Besonderheiten zu beachten: 1. Beim Rücksprung in das Betriebssystem QDOS muß D0 gelöscht sein; das bewirkt der Befehl MOVEQ #0,D0 mit Opcode 7000. 2. Hierdurch verschiebt sich die Zieladresse um zwei zu den höheren Adressen hin. 3. Wenn einmal etwas schiefläuft und der Ablauf gewaltsam unterbrochen werden muß, wird das Programm nicht spurlos getilgt – auch nicht, wenn NEW gegeben wurde. Bei erneutem Start – oder nach neuem Laden – wird ein anderer Bereich angezeigt. Man muß das Maschinenprogramm neu eingeben und sozusagen noch vor dem Bildschirm anpassen.

```
03F000  23FC 2807 1986 0003 F00E  MOVE.L  #$28071986,$3F00E
03F00A  7000                      MOVEQ   #0,D0
03F00C  4E75                      RTS
```

P.2.3 Das erste Programm für den QL

Abschließend noch eine Empfehlung: Experimentieren Sie mit diesen (und allen anderen) Programmen; verändern Sie mit Vorbedacht die Zieladresse; übertragen Sie auch andere Daten!

3
Speicherplatzunabhängige Adressierung (I)

Beim QL war zu beobachten, daß ein und dasselbe BASIC-Programm einmal hier und einmal dort im Speicher sitzen kann. Das ist kein Fehler. Im Gegenteil: Es gehört zu den Vorzügen des M68000, daß sich für ihn mühelos «relokatible» (= im Speicher – nahezu – beliebig verschiebbare) Programme schreiben lassen. So ist es u.a. möglich, ein nur einmal formuliertes Unterprogramm mal in dieses und mal in jenes Programm *unverändert* einzubauen.

3.1 Die Berechnung der Adreßdistanz

Obgleich in der Praxis der Assembler diese Aufgabe in wohl weniger als einer zehntel Sekunde beiläufig mit erledigt, sollte man das Verfahren und auch das Wichtigste über die Vorgänge im Prozessor kennen; das ist auch für das Verständnis anderer Befehle von Nutzen. Die Ausführungen beziehen sich auf das folgende Programmbeispiel.

Die Bearbeitung des LEA-Befehls beginnt damit, daß der Prozessor den Opcode (hier 41FA) aus dem Speicher holt. Dabei zeigt ein Register – der Programmzähler (program counter, PC) – auf die Speicherstelle, in der dieser Opcode steht; also PC = 009C02. Nachdem diese Arbeit getan ist, wird der Programmzähler auf das nächste Datenwort im Speicher gerichtet, also um zwei erhöht: PC = 009C04. Nach der internen Decodierung berechnet die CPU den Wert, mit dem A0 zu laden ist, indem er zum PC vorzeichenrichtig die Adreßdistanz addiert: A0 = 009C04 + 0006 = 009C0A. Der nächste Befehl legt schließlich den Inhalt von D0 in der von A0 bezeichneten Speicherstelle ab. Die 16-bit-Adreßdistanz errechnet sich umgekehrt als Differenz zwischen Zieladresse 009C0A

und Programmzählerstand 009C04 (−)

zu 0006 (=).

A.3.1: Bitte berechnen Sie folgende Adreßdistanzen:
a) PC = 023456, Zieladresse = 02468A;
b) PC = 01DCBA, Zieladresse = 01EFEC;
c) PC = 0350CE, Zieladresse = 035370;
d) PC = 015234, Zieladresse = 01D234;
e) LEA wird mit PC = 015234 erreicht, Ziel wie d).

3.1.1 Ein Beispiel für den NDR-Computer

```
009C00  703B        MOVEQ   #$3B,D0
009C02  41FA 0006   LEA     $0006(PC),A0
009C06  2080        MOVE.L  D0,(A0)
009C08  4E75        RTS
```

P.3.2 Ein relokatierbares Programm für den NDR-Computer

Der Befehl MOVEQ #$3B,D0 (move quick, bewege schnell) lädt das D0-Register mit der unmittelbar gegebenen 8-bit-Konstante $3B. LEA bedeutet «Lade eine effektive Adresse»; hier wird A0 mit der Adresse des auf das Programmende folgenden Speicherplatzes geladen (also A0 = $9C0A). Diese Adresse wird programmzählerrelativ ermittelt, worauf wir schnell zurückkommen werden. Der dritte Befehl transferiert den gesamten 32-bit-Inhalt von D0 in die durch A0 bezeichnete Adresse; man darf gespannt sein, was dieses Langwort enthält. Der Rücksprung ins Betriebssystem ist bereits bekannt.

A.3.2: Bitte versuchen Sie, folgende Fragen zu beantworten:
a) Nach welcher Adressierungsart wurde im dritten Befehl das Ziel bezeichnet?
b) Ermitteln Sie den vom dritten Befehl übertragenen Inhalt von D0.
c) Versuchen Sie, den ersten Opcode so zu verändern, daß die Konstante $A3 geladen wird. c1) Wie lautet der Opcode? c2) Was kommt an?

Sie finden die Lösung(en) dieser Aufgabe(n) im Anhang C.

3.1.2 Ein Beispiel für den Amiga

```
035C00  48E7 8080  MOVEM.L D0/A0,-(A7)
035C04  703B       MOVEQ   #$3B,D0
035C06  41FA 000A  LEA     $000A(PC),A0
035C0A  2080       MOVE.L  D0,(A0)
035C0C  4CDF 0101  MOVEM.L (A7)+,D0/A0
035C10  4E75       RTS
```

P.3.1 Ein relokatierbares Programm für den Amiga

Der Verfasser kennt bis zur Stunde noch keine explizite Angabe darüber, welche Register bei Anwendung des CALL-Befehls geschützt sind. Das Beispiel auf den Seiten 6 bis 10 des Amiga-BASIC-Handbuchs legt nahe, die Inhalte der beteiligten Register vor ihrer Benutzung an einen sicheren Ort zu «retten» und vor Rückkehr nach BASIC wiederherzustellen. Anderenfalls könnte es geschehen, daß der Computer aus der Anwenderroutine mit falschen Voraussetzungen (bzw. Parametern, Systemvariablen usw.) zurückkehrt; ein Systemabsturz wäre die wahrscheinliche Folge. Für die sichere Zwischenspeicherung der fraglichen Registerinhalte sorgen die beiden MOVEM-Befehle. Der Leser wird gebeten, sie an dieser Stelle ohne weitere Erklärung als Vorgabe zu akzeptieren. Das zweite MOVEM verändert die Adreßdistanz im Vergleich zu P.3.2. Es bleibt dem Leser überlassen, die Distanz nachzurechnen.

```
03F800  703B       MOVEQ   #$3B,D0
03F802  41FA 0008  LEA     $0008(PC),A0
03F806  2080       MOVE.L  D0,(A0)
03F808  7000       MOVEQ   #0,D0
03F80A  4E75       RTS
```

P.3.3 Ein relokatierbares Programm für den QL

3.1.3 Ein Beispiel für den QL

Vor dem Rücksprung muß – wie bekannt – lediglich D0 gelöscht werden. Das MOVEQ führt hier auf eine wiederum veränderte Distanz.

3.2 Das Programm wird reloziert

Die in diesem Abschnitt vorgestellte Adressierung benötigt zwar – etwa im Vergleich von P.2.1 mit P.3.2 – zwei Programmzeilen mehr, aber zwei Bytes weniger. Entscheidend ist, daß sich die so erstellten Programme an (nahezu) beliebiger Stelle im Speicher unterbringen lassen und dort ohne jede Anpassung laufen. Sie sollten das unbedingt ausprobieren. Tippen Sie z.B. P.3.3 statt ab 03F800 ab 03F810 oder ab 03F828 ein – vorausgesetzt, daß Ihnen das Monitorprogramm diesen Bereich reservierte. Provozieren Sie auch ruhig einmal einen Programm- oder gar Systemabsturz, indem Sie das Programm mit einer ungeraden Adresse beginnend eintippen!

4
Speicherplatzunabhängige Adressierung (II)

Bevor im nächsten Kapitel in die mühselige Codierung von Hand eingeführt wird, will dieses die Leistungsfähigkeit des M68000 – insbesondere seiner indirekten Adressierungsarten – noch einmal verdeutlichen; es ist sozusagen ein Kompliment.

4.1 Mehrfache Ablage im Speicher mit Postinkrement

```
009C00   70AB        MOVEQ    #$AB,D0
009C02   41FA 000A   LEA      $000A(PC),A0
009C06   20C0        MOVE.L   D0,(A0)+
009C08   30C0        MOVE.W   D0,(A0)+
009C0A   10C0        MOVE.B   D0,(A0)+
009C0C   4E75        RTS
```

P.4.1 Mehrfache Ablage mit dem NDR-Computer

Entscheidend ist hier die Adressierungsart «Adreßregister indirekt mit Postinkrement". Der erste der drei Ablagebefehle kopiert den gesamten Inhalt von D0 in den von A0 angezeigten Speicherbereich; dann wird der Inhalt von A0 – .L entsprechend – um 4 erhöht. Am Ende der Ausführung des zweiten Ablagebefehls wird A0 um 2 erhöht; am Ende des dritten ganz entsprechend um 1.

A.4.1: Sagen Sie voraus, wie der Ablagebereich nach dem Programmablauf aussieht, und begründen Sie.

A.4.2: Schreiben Sie die P.4.1 entsprechenden Programme
a) für den QL;
b) für den Amiga.

A.4.3: Kehren Sie die Reihenfolge der Ablagebefehle um, probieren Sie, beobachten Sie, deuten Sie!

A.4.4: Sie werden zu einer Gratwanderung eingeladen:
a) Richten Sie A0 auf den Programmanfang, so daß sich P.4.1 selbst überschreibt.
b) Sagen Sie den Speicherinhalt voraus.
c) Wie oft darf MOVE.B als dritter, vierter usw. Befehl eingebaut werden, ohne daß der Rücksprung ins Betriebssystem überschrieben wird?
d) Wie oft dürfte von Anfang an und ohne Gefährdung des Rücksprungs MOVE.W gesetzt werden?
e) MOVE.L? – Zur Nachahmung auch QL- und Amiga-Besitzern empfohlen.

4.2 Mit Predekrement

```
009C00  203C 1234 5678  MOVE.L   #$12345678,D0
009C06  41FA 0012       LEA      $12(PC),A0
009C0A  2100            MOVE.L   D0,-(A0)
009C0C  3100            MOVE.W   D0,-(A0)
009C0E  1100            MOVE.B   D0,-(A0)
009C10  4E75            RTS
009C12  4E75            RTS
```

P.4.2 Mehrfache Ablage mit Predekrement

A.4.5: Statt einer unnötigen Erklärung einige Aufgaben:
a) Sagen Sie den Speicherinhalt ab 009C10 voraus.
b) Wo beginnt der erste Eintrag?
c) Wo ist der letzte eingeschrieben?
d) Wozu das zweite RTS?

4.3 Ein Programm kopiert sich selbst

```
035C00  48E7 FFFE  MOVEM.L D0-D7/A0-A6,-(A7)
035C04  41FA FFFA  LEA     $FFFA(PC),A0
035C08  43FA 001A  LEA     $001A(PC),A1
035C0C  22D8       MOVE.L  (A0)+,(A1)+
035C0E  22D8       MOVE.L  (A0)+,(A1)+
035C10  22D8       MOVE.L  (A0)+,(A1)+
035C12  22D8       MOVE.L  (A0)+,(A1)+
035C14  22D8       MOVE.L  (A0)+,(A1)+
035C16  22D8       MOVE.L  (A0)+,(A1)+
035C18  22D8       MOVE.L  (A0)+,(A1)+
035C1A  22D8       MOVE.L  (A0)+,(A1)+
035C1C  22D8       MOVE.L  (A0)+,(A1)+
035C1E  4CDF 7FFF  MOVEM.L (A7)+,D0-D7/A0-A6
035C22  4E75       RTS
```

P.4.3 Ein sich selbst kopierendes Programm (Amiga)

Unser Programm tut zunächst einmal des Guten zuviel, indem es statt der nur zwei benutzten (Adreß-)Register «alle» 15 Register (A7 ist einer besonderen Verwendung vorbehalten) vorübergehend an sicherer Stelle zwischenspeichert. Dieser Weg wurde gewählt, um dem Amiga-Besitzer ein Allround-Werkzeug in die Hand zu geben, das – bis zur Besprechung des MOVEM-Befehls – immer paßt.

A.4.6: Auch hier sollen Fragen zum Verständnis führen:

a) Auf welche Stelle wird A0 gerichtet?
b) Und A1?
c) Was bewirkt der erste MOVE.L-Befehl?
d) Der zweite?
e) Weshalb wird der Befehl gerade neunmal eingesetzt?
f) Worin unterscheidet sich die Kopie vom Original?
g) Weshalb schadet das nichts?
h) Welches ist die Startadresse des Tochterprogramms?
i) Kann man es gefahrlos starten?
j) Was bewirkt es?

k) Ist auch das Tochterprogramm der F2-Generation lauffähig?
l) Wie oft läßt sich das Spielchen wiederholen?

A.4.7: Schreiben Sie das Programm
a) für den QL,
b) für den NDR-Computer um.

5
Vom Mnemonik zum Opcode

Das Grundschema, nach dem die Opcodes des M68000 aufgebaut sind, zeigt Bild 5.1.

Wie zu erkennen ist, sind i.a. ein Nibble und vier Tripel zu einem Wort zusammenzusetzen. Dabei bedeutet

zzz	das führende Nibble,
rrr	eine Registernummer und
mmm	«mode» eine Adressierungsart.

Beachten Sie, daß hier im Codierungsschema Quelle und Ziel in umgekehrter Reihenfolge auftreten!

Der aufmerksame Leser wird bemerkt haben, daß drei Bits nur acht verschiedene Werte darstellen können, während es aber zwölf (reguläre) verschiedene Adressierungsarten gibt. Dieser scheinbare Widerspruch wird sich klären.

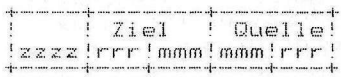


Bild 5.1

	Dn	An	(An)	(An)+	-(An)	d(An)	d(An,Rx.X)	$xxxx	$xxxxxxxx	d(PC)	d(PC,Rx.X)	#
Quelle:	X	X	X	X	X	X	X	X	X	X	X	X
Ziel:	X		X	X	X	X	X	X	X			

Bild 5.2 Die für den Befehl MOVE zugelassenen Adressierungsarten

5.1 Die Codierung des Befehls MOVE

Der Befehl MOVE.X <von>,<nach> bewegt ein Datum von der Quelle zum Ziel. Bild 5.2 zeigt, welche Adressierungsarten hierfür zugelassen sind. Das Codierungs-Grundschema (Bild 5.3) ist hier etwas abgewandelt:

gg («Größe») bezieht sich auf das Datenformat .X:
01 bedeutet hier .B,
10 bedeutet hier .L, und
11 bedeutet hier .W. Die Bitkombination 00 bedeutet hier zum mindesten nicht MOVE.
mmm bezeichnet den Adressierungsmodus. Die ersten 7 Modi sind einfach der Reihe nach durchnumeriert.

Dann übernimmt – die Zahlen in Bild 5.4 sind binär zu schreiben – das Tripel «Modus» die Funktion einer Vorwahlnummer, und die Tripel «R'Nr.» 000 ... 100 übernehmen die weitere Fallunterscheidung.

Bild 5.3 Das Codierungsschema für MOVE

	Dn	An	(An)	(An)+	-(An)	d(An)	d(An,Rx.X)	$xxxx	$xxxxxxxx	d(PC)	d(PC,Rx.X)	#
Modus:	0	1	2	3	4	5	6	7	7	7	7	7
R'Nr.:								0	1	2	3	4

Bild 5.4 Die Codierung der Adressierungsarten

5.2 Assemblieren und Disassemblieren (I)

Die Übersetzung eines mnemonisch geschriebenen Befehls in die Maschinensprache (Opcode bzw. Objektcode) heißt «Assemblieren»; diese Aufgabe übernimmt in der Praxis ein besonderes Assemblierungsprogramm, kurz «Assembler» genannt. Disassemblieren ist die Umkehrung, die Rückübersetzung des Op(erations)codes in die mnemonische Formulierung. Wir benutzen diesen Weg zum ersten Verständnis des in Abschnitt 5.1 Dargestellten:

In P.4.1 und folgenden wurde der Opcode 30C0 verwendet; er lautet binär notiert
0011 0000 1100 0000 oder Bild 5.3 entsprechend umgruppiert
00 11 000 011 000 000.

Da der Opcode mit 00, aber nicht mit 0000 beginnt, handelt es sich um einen MOVE-Befehl. gg = 11 codiert das Datenformat .W.

Das Ziel wird beschrieben durch mmm = 011; es ist also durch die Adressierungsart (An)+ bezeichnet; dann ist rrr = 000 (erstes Tripel) die Registernummer des Zielregisters.

Die Quelle wird beschrieben durch mmm = 000, also als «Datenregister direkt»; dann ist rrr = 000 (letztes Tripel) die Nummer des Quell-Datenregisters.

Damit haben wir den Mnemonik MOVE.W D0,(A0)+ zurückgewonnen.

A.5.1: Bitte codieren Sie:

a) MOVE.B D1, D2
b) MOVE.W D3, A4
c) MOVE.L A4, D3
d) MOVE.B D5,(A6)
e) MOVE.B $23(PC),(A0)
f) MOVE.W $82(PC),D7
g) MOVE.L D4,$81(PC)

5.3 Die Codierung des Befehls MOVEQ

Der Befehl MOVEQ #Konst,Dn lädt das adressierte Datenregister mit einer vorzeichenbehafteten 8-bit-Zahl (und füllt den Inhalt des Datenregisters – wie in P.3.1 und P.4.1 zu sehen war – vorzeichenrichtig auf). Das geht schnell (quick), weil Opcode und Operand eng verschmolzen sind, so daß nur ein Speicherzugriff erforderlich ist. Für Quelle und Ziel ist nur je eine Adressierungsart zugelassen. Bild 5.5 zeigt das Codierungsschema.

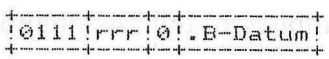


Bild 5.5 Die Codierung von MOVEQ

A.5.2: Bitte codieren Sie:

a) MOVEQ #$12,D0
b) MOVEQ #$34,D1
c) MOVEQ #56,D2
d) MOVEQ #−76,D3
e) MOVEQ #−$76,D4
f) MOVEQ #$9A,D5
g) MOVEQ #$BCDE,D7

5.4 Die Codierung der Adressierungsart d(An,Rx.X)

Bild 5.6 zeigt das Codierungsschema.

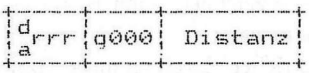


Bild 5.6 Die Codierung von d(An,Rx.X)

Bei der Adressierungsart «Adreßregister indirekt mit Index und Adreßdistanz» setzt sich die effektive Adresse additiv aus dem Inhalt von An, dem Inhalt von Rx und der vorzeichenbehafteten 8-bit-Distanz d zusammen. Wir kleiden die Beispiele in eine Aufgabe:

A.5.3: Angenommen, A4 = 03F856, A5 = 05CDEE und D3 = 456789AB. Berechnen Sie die sich aus folgenden Befehlsbestandteilen ergebenden effektiven Adressen:

a) $10(A4,A5.L)
b) $9C(A4,A5.L)
c) $23(A4,A5.W)
d) $67(A5,D3.L)
e) $67(A5,D3.W)
f) $AB(A5,D3.W)

Zur Codierung:
d/a = 0 besagt, daß R ein Datenregister ist;
d/a = 1 besagt, daß R ein Adreßregister ist.
rrr ist die Nummer dieses Indexregisters.

Für das Format des Indexes sind .W und .L zugelassen;
g = 0 steht für .W,
g = 1 steht für .L.
ddddddddd ist die 8-bit-Adreßdistanz d.

A.5.4: Codieren Sie die sechs Beispiele aus A.5.3.

5.5 Ein Experimentierprogramm zu d(An,Rx.X)

Nach dieser harten Kost ein Dessert:

```
009C00  48E7 FFFE          MOVEM.L D0-D7/A0-A6,-(A7)
009C04  287C 0003 F856     MOVEA.L #$03F856,A4
009C0A  2A7C 0005 CDEE     MOVEA.L #$05CDEE,A5
009C10  263C 4567 89AB     MOVE.L  #$456789AB,D3
009C16  4DFA 002C          LEA     BUFFER(PC),A6
009C1A  41F4 D810          LEA     $10(A4,A5.L),A0
009C1E  2CC8               MOVE.L  A0,(A6)+
009C20  41F4 D89C          LEA     $9C(A4,A5.L),A0
009C24  2CC8               MOVE.L  A0,(A6)+
009C26  41F4 D023          LEA     $23(A4,A5.W),A0
009C2A  2CC8               MOVE.L  A0,(A6)+
009C2C  41F5 3867          LEA     $67(A5,D3.L),A0
009C30  2CC8               MOVE.L  A0,(A6)+
009C32  41F5 3067          LEA     $67(A5,D3.W),A0
009C36  2CC8               MOVE.L  A0,(A6)+
009C38  41F5 30AB          LEA     $AB(A5,D3.W),A0
009C3C  2CC8               MOVE.L  A0,(A6)+
009C3E  4CDF 7FFF          MOVEM.L (A7)+,D0-D7/A0-A6
009C42  4E75               RTS
009C44                BUFFER: DS.L    6
009C5C                     END
```

P.5.1 Das Experimentierprogramm zu d(An,Rx.X) für den NDR-Computer

Obgleich beim NDR-Computer nur A5 und A7 nicht verändert werden dürfen, tun wir wieder einmal des Guten zuviel und setzen den MOVEM-Befehl zur Rettung aller anderen Register – wie in Abschnitt 4.3 beschrieben – ein.

Wenn ein Adreßregister Ziel eines Datentransfers ist, wird MOVE zu MOVEA. Hier wird z.B. A4 mit der als Konstante («#») gegebenen Adresse $03F856 geladen. MOVEA.L $03F856,A4 (Adressierungsart absolut) hätte dagegen das in $03F856 beginnende Langwort als Adresse nach A4 geladen.

LEA errechnet – wie bereits mitgeteilt – eine effektive Adresse und lädt sie in ein Adreßregister. Dieser Befehl löst uns hier A.5.3.

Schließlich sei noch darauf hingewiesen, daß hier erstmals weitere Elemente der Assemblersprache benutzt wurden:

Der Asteriskus «*» zeigt einen Kommentar an, der vom Assembler ignoriert wird, aber dennoch im Assemblerprotokoll erscheint. Er entspricht dem BASIC-REM. Das END zeigt dem Assembler an, wo seine Arbeit zu beenden ist.

Sehr hilfreich sind die symbolischen Adressen (Labels, Marken), ein Mittelding zwischen Variable und Konstante: Anfangs sind sie leer. Ein später zugewiesener Wert bleibt unveränderlich erhalten, kann jederzeit abgerufen und in Adreß- und Distanzberechnungen einbezogen (aber auch zur Definition von Datenkonstanten benutzt) werden. Im ersten Durchlauf stellt der Assembler fest, daß z.B. in P.5.1 dem Label BUFFER die Adresse $009C44 zuzuweisen ist. Er notiert das in seiner Symboltafel. Im zweiten Durchlauf (so beim Zwei-Pass-Assembler) errechnet er den zum fünften Befehl des Programms gehörenden Offset (Adreßdistanz) $009C44 – $009C18 = $002C.

Die Assemblerdirektive DS (define storage, definiere Speicher) reserviert (in unserem Zusammenhang unnötig) für 6 Langwörter 24 Bytes und zwingt den Assembler zur Anzeige der Anfangsadresse des Puffers.

5.6 Zur Quelle-Ziel-Terminologie

In 5.1 wurden die Begriffe Quelle und Ziel benutzt; sie bezeichnen nicht nur beim Datentransfer Herkunft und Bestimmung, sondern auch bei vielen Verknüpfungen, z.B. arithmetischen Operationen. Soll etwa $12 + $34 = $46 berechnet werden, so ist einer der beiden Summanden in der Quelle bereitzustellen, der andere im Ziel. Das Resultat ist nach der Addition im Ziel zu finden.

6
Zählerkontrollierte Programmschleifen

Ein Looping ist eine Kunstflugfigur, bei der das Flugzeug ein und dieselbe Stelle zweimal passiert; beim ersten Mal macht es einen Überschlag (Schleife nach oben), beim zweiten fliegt es geradeaus weiter. Mehrere Überschläge sind denkbar. Das ist auch die Struktur einer Programmschleife, bei der eine Gruppe von Befehlen ein- oder mehrmals durchlaufen wird. In diesem Kapitel wird die Anzahl der Durchläufe in einem Datenregister vorgegeben und in diesem kontrolliert (Bild 6.1).

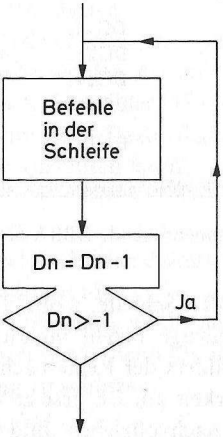


Bild 6.1 Struktur einer DBRA-Schleife

Der M68000 erledigt diese Kontrolle in einem einzigen Befehl. Wir werden sehen, daß er sogar noch eine weitere Bedingung in einem Gang überprüfen kann.

6.1 DBRA-Schleifen

Der Befehl DBRA Dn,Label besagt im Klartext: «Dekrementiere das Register Dn und verzweige so lange zum Label, bis das Zählregister Dn den Wert −1 enthält» (verzweigen = to branch). DBRA ist eine von Assemblern nach Motorola-Norm tolerierte Zweitbezeichnung des Befehls DBF, der in einem allgemeineren Zusammenhang besprochen wird.

```
035C00  48E7 FFFE          MOVEM.L  D0-D7/A0-A6,-(A7)
035C04  47FA 0010          LEA      BUFF(PC),A3
035C08  7A0B               MOVEQ    #$0B,D5

035C0A  16C5         LOOP: MOVE.B   D5,(A3)+
035C0C  51CD FFFC          DBRA     D5,LOOP

035C10  4CDF 7FFF          MOVEM.L  (A7)+,D0-D7/A0-A6
035C14  4E75               RTS

035C16  EEEE EEEE          DC.L     $EEEEEEEE
035C1A  EEEE EEEE          DC.L     $EEEEEEEE
035C1E  EEEE EEEE          DC.L     $EEEEEEEE
035C22  EEEE EEEE          DC.L     $EEEEEEEE

                           END
```

P.6.1 Selbstdokumentierende DBRA-Schleife (Amiga)

P.6.1 enthält eine solche DBRA-Schleife, wobei D5 als Schleifenzähler (im .W-Format) fungiert. Der einzige Befehl innerhalb der Schleife legt das jeweilige LSB des Schleifenzählers der Reihe nach im Puffer ab, der vorher zur Verbesserung der Lesbarkeit auf EE gesetzt wurde. So kann man den Ablauf recht gut gedanklich nachvollziehen (Bild 6.2).

```
035C00  48E7 FFFE 47FA 0010 7A0B 16C5 51CD FFFC
035C10  4CDF 7FFF 4E75 0B0A 0908 0706 0504 0302
035C20  0100 EEEE ....
```

Bild 6.2 Hexdump nach Programmablauf

Wir benutzen nun den DBRA-Befehl in einer Variante von P.4.3:

```
035C00  48E7 FFFE            MOVEM.L D0-D7/A0-A6,-(A7)
035C04  4DFA 0012            LEA     BUFFER(PC),A6
035C08  7A0B                 MOVEQ   #12-1,D5
035C0A  3CEE FFE8  LOOP:     MOVE.W  -24(A6),(A6)+
035C0E  51CD FFFA            DBRA    D5,LOOP
035C12  4CDF 7FFF            MOVEM.L (A7)+,D0-D7/A0-A6
035C16  4E75                 RTS
                   BUFFER:   DS.W    12
```

P.6.2 Noch ein Selbstkopierer (für den Amiga)

Mit der ORG-Direktive wird der Programmzähler hier auf $35C00 gesetzt; der RDK-Assembler hilft dem Amiga!

Da das Programm 12 Wörter lang ist, muß bei einem .W-Transfer die Schleife zwölfmal durchlaufen werden. Da aber beim Zählerstand 0 noch ein Durchlauf erfolgt, ist der Schleifenzähler D5 mit $B zu laden. Beachten Sie, daß der Assembler nicht nur auch Dezimalzahlen annimmt, sondern auch noch einfache Rechnungen ausführen kann!

A.6.1:

a) Geben Sie genau an, welche beiden Adressierungsarten im wesentlichen Transfer-Befehl verwendet wurden.

b) Schreiben Sie das Programm für den QL mit Startadresse 3F800 so um, daß byteweise transferiert wird. (Beim QL darf A6 nicht benutzt werden – oder ist bei Benutzung zu schützen.)

6.2 Schleifen mit Zweitbedingung (DBcc)

Die abstrakte Grundform des DBRA-Befehls ist DBcc, was bedeutet: «Dekrementiere und verzweige, wenn der Zählerstand von −1 verschieden ist. Wenn aber die Bedingung cc (condition code) erfüllt ist, dann weiter, ohne zu dekrementieren».

In P.6.3 soll der Computer selber aufpassen, wann er das Programm fertig kopiert hat; warum sollen wir denn erst die Programmlänge auszählen?

```
009C00  41FA FFFE  START:  LEA      START(PC),A0
009C04  43FA 0014          LEA      BUFFER(PC),A1
009C08  7263               MOVEQ    #99,D1
009C0A  3018       LOOP:   MOVE.W   (A0)+,D0
009C0C  32C0               MOVE.W   D0,(A1)+
009C0E  0C40 4E75          CMPI.W   #$4E75,D0
009C12  57C9 FFF6          DBEQ     D1,LOOP
009C16  2288               MOVE.L   A0,(A1)
009C18  4E75               RTS
009C1A             BUFFER: DS.W     13
```

P.6.3 Selbstkopierer mit Selbstkontrolle (NDR)

Es ist nicht schwer zu erkennen, daß hier die zu transferierenden Daten zunächst nach D0 geholt und dann aus D0 wieder abgelegt werden. Das ermöglicht eine einfache Kontrolle der übertragenen Daten: Der Befehl CMPI (compare immediate) vergleicht den Inhalt von D0 mit der unmittelbar (immediate) gegebenen Konstante $4E75. Das Ergebnis dieses Vergleichs wird in vier Stellen des Condition Code Registers (CCR) zwischengespeichert. Der folgende Befehl DBEQ (Decrement and Branch until EQual) prüft, ob im CCR Gleichheit festgehalten wurde. Ist das der Fall, geht es ohne zu dekrementieren geradlinig weiter. Bei Ungleichheit wird dekrementiert und, wenn der Schleifenzähler noch nicht -1 erreichte, zum markierten Ziel zurückgesprungen.

An sich überflüssig, aber doch nützlich zur Kontrolle, wird zum Schluß noch der momentane Inhalt von A0 im Puffer abgelegt.

A.6.2:

a) Für wie viele Durchläufe wurde der Schleifenzähler präpariert?
b) Wie viele Durchläufe fanden tatsächlich statt?
c) Sagen Sie den Inhalt von A0 bei Ablage im Puffer voraus.
d) Ändern Sie das Programm so ab, daß auch der Endstand von D0 abgelesen werden kann.

7
Vom Condition Code Register (CCR)

Das CCR ist das LSB des Statusregisters (SR), das erst viel später besprochen werden soll. Nur fünf (manchmal nur vier) seiner Bits werden benutzt (Bild 7.1).

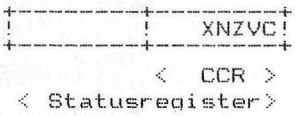


Bild 7.1 Das Condition Code Register (CCR)

Viele Befehle beeinflussen diese Bits – je nachdem, ob bestimmte Ereignisse («Ergebnis null», «Ergebnis negativ», «Übertrag» usw.) eintraten. Zu ihnen gehört auch CMPI (siehe P.6.3). Die Bits des CCR werden auch als Flags ((Warn-)Flagge) bezeichnet.

7.1 Die Flags des CCR (I)

C-Flag (carry) wird gesetzt, wenn bei einer Operation ein Übertrag auftritt. Beispiel: $80 + $80 = $00. Hierbei ist eine 1 zum nächsthöherwertigen Nibble zu übertragen.

V-Flag: $40 ist eine positive Zahl. $40 + $40 = $80. Hier ist als Summe zweier positiver Zahlen eine formal negative Summe (das höchstwertige Bit ist gesetzt) entstanden, was man als Overflow (Überlauf) bezeichnet. Hierauf weist V-Flag hin.

Z-Flag: Zeroflag (Nullflag) weist auf das Ergebnis null hin. Da bei einem Vergleich die beiden zu vergleichenden Summanden intern subtrahiert werden, zeigt Z-Flag hier auch Gleichheit an.

N-Flag zeigt negative Ergebnisse an.

X-Flag (extend bit) wird i. a. wie C-Flag gesetzt. Damit wird es möglich, das eine der beiden in der nächsten Operation mitzuverwenden und das andere für eine spätere aufzubewahren.

7.2 Beobachtung des CCR

```
000'  48E7 FFFE            MOVEM.L D0-D7/A0-A6,-(A7):
004'  41FA 0014            LEA     BUFFER(PC),A0
008'  303C 1234            MOVE.W  #$1234,D0
00C'  323C 5678            MOVE.W  #$5678,D1
010'  B041                 CMP.W   D1,D0
012'  40D0                 MOVE    SR,(A0)
014'  4CDF 7FFF            MOVEM.L (A7)+,D0-D7/A0-A6
018'  4E75                 RTS
01A'              BUFFER:  DS.W    1
```

P.7.1 Die Beobachtung des CCR

Zwei Befehle sind zu P.7.1 zu besprechen:

CMP.W D1,D0 subtrahiert intern (d. h. ohne Veränderung der beiden Operanden) D1 von D0. Ist D1 > D0, dann ist das Ergebnis negativ; außerdem findet ein Übertrag statt.

MOVE SR,<ea> kopiert den Inhalt des Statusregisters, von dem hier nur das LSB interessiert, zur effektiven Adresse, hier also zum Puffer.

Das Listing zeigt keine absoluten Adressen – wozu auch? Man braucht ja gar nicht zu wissen, wo das Programm im Speicher untergebracht wird.

Um mit dem Programm bequem experimentieren zu können, wird es in ein Hostprogramm (Wirtsprogramm), manchmal auch als Lader bezeichnet, eingebunden:

Es wird nicht schwerfallen, es auf andere Computer zu übertragen. Die Zahlen, mit denen man gern experimentieren möchte, werden direkt ins Programm eingePOKEt, und das interessierende Resultat aus dem Puffer herausgePEEKt. Abschließend werden die Symbole der gesetzten Bits des CCR angezeigt; für die anderen erscheinen Punkte als Platzhalter. Sie werden

```
CLS: PRINT "Die Beobachtung des CCR"
code$=STRING$(768,0): anfad=SADD(code$): ccr$="XNZVC"
DATA "48E7","FFFE","41FA","0014","303C","1234","323C"
DATA "5678","B041","40D0","4CDF","7FFF","4E75"
RESTORE: FOR i=0 TO 12
         READ h$: POKEW anfad+2*I,VAL("&H"+h$):NEXT i
loop:
INPUT"1. Zahl = ";h$: POKEW anfad+10,VAL("&H"+h$)
INPUT"2. Zahl = ";h$: POKEW anfad+14,VAL("&H"+h$)
         CALL anfad: ccr=PEEK(anfad+27)
         PRINT"  CCR = ";: FOR i=1 TO 5: ps$="."
         IF ccr AND 2^(5-i) THEN ps$=MID$(ccr$,i,1)
         PRINT ps$;: NEXT i
         PRINT: PRINT: GOTO loop
```

P.7.1H Das Hostprogramm für den Amiga

sich vielleicht über einige Ergebnisse wundern, doch ein Blick ins Handbuch lehrt, daß CMP das X-Flag nicht beeinflußt. Wenn Sie B041 durch D001 (ADD) oder 9001 (SUB) ersetzen, wird auch das X-Flag in den Ablauf einbezogen.

7.3 Flagkombinationen und Bedingungen

Wir greifen zunächst ein ganz konkretes Beispiel heraus und stellen von zwei Zahlen fest, ob z0 > z1. Dieses «größer als» darf nicht unreflektiert mit «higher» (Bedingungsschlüssel HI) übersetzt werden. Zunächst passen wir P.7.1 bzw. P.7.1H ein wenig an die Fragestellung an: Übersichtlichkeitshalber werden die Zahlen auf 8-bit-Format begrenzt; dementsprechend ist für B041 der Opcode B001 (CMP.B D1,D0) einzusetzen. In das Hostprogramm wird noch eingefügt IF (ccr AND 5)=0 THEN PRINT " HI".

Die Flagkombination 5, binär 0101 markiert Z- und C-Flag. Das leuchtet ein, denn wenn z0 > z1, dann ist das Ergebnis 1. nicht null, und 2. findet bei der internen Subtraktion z0 – z1 kein Übertrag statt; Zero- und Carryflag sind also beide gelöscht. Man kann das mit den Symbolen der Aussagenlogik bzw. der Booleschen Algebra auch so schreiben:

$$\overline{C} \wedge \overline{Z} = 1$$

oder etwas weniger abstrakt: HI liegt vor, wenn

$$(C = 0) \text{ und } (Z = 0).$$

Mit dem abgewandelten Programm erhält man unproblematische Ergebnisse wie FD < FF, was in beiden Sichtweisen richtig ist: $-3 < -1$ für vorzeichenbehaftete Zahlen und $253 < 255$ unter Zugrundelegung vorzeichenloser Zahlen (unsigned numbers). Ergebnisse wie 7F < 82 oder 00 < FE zeigen aber, daß die Bedingung HI nur für vorzeichenlose Zahlen gilt.

Das Rechnen mit vorzeichenbehafteten Zahlen (signed numbers) wird auch Zweierkomplementarithmetik genannt. ((−1) + (+1) = 0 sieht in binärer Notierung so aus: 11111111 + 00000001 = (1)00000000; in jeder Binärstelle entsteht bei der Addition 10 also 2. Der eine Teilsummand ergänzt den anderen zu 2, er ist also sein Zweierkomplement). Für diese Interpretation gilt «GT» (greater than) statt «HI». Die zugehörige Flagkombination ist

$$(N \wedge V \wedge \overline{Z}) \vee (\overline{N} \wedge \overline{V} \wedge \overline{Z}) = 1$$

oder auch so:

$$((N = 1) \text{ und } (V = 1) \text{ und } (Z = 0))$$

oder

$$((N = 0) \text{ und } (V = 0) \text{ und } (Z = 0)).$$

Die entsprechende Änderung im Experimentierprogramm wäre:
IF (ccr AND 14)=10 OR (ccr AND 14)=0 THEN PRINT "GT".
Jetzt wird – in bezug auf signed numbers – richtig festgestellt: 7F > 82 bzw. 00 > FE.

7.4 Die sechzehn Bedingungsschlüssel

Die einzelnen Flagkombinationen kann man im Handbuch nachlesen. Für das praktische Programmieren reicht die folgende Übersicht aus:

0000 T Die Bedingung ist immer erfüllt (true).
0001 F Die Bedingung ist nie erfüllt (false).
0010 HI Das Ergebnis ist «höher» (higher).

0011 LS Das Ergebnis ist kleiner gleich (less or same).
0100 CC Carryflag ist gelöscht (carry clear).
0101 CS Carryflag ist gesetzt (carry set).
0110 NE Ungleich (null) (not equal).
0111 EQ Das Ergebnis ist gleich (null) (equal).
1010 PL Das Ergebnis ist positiv (plus).
1011 MI Das Ergebnis ist negativ (minus).

Und für signed numbers (Zweierkomplementarithmetik):
1000 VC Kein Überlauf (overflow clear).
1001 VS Überlauf (overflow set).
1100 GE Größer oder gleich (null) (greater or equal).
1101 LT Kleiner als (less than).
1110 GT Größer als (greater than).
1111 LE Kleiner gleich (null) (less or equal).

Die vorangestellten Binärzahlen geben die Codierung der Bedingung cc innerhalb des Opcodes wieder.

7.5 Die Codierung des Befehls DBcc Dn,Label

Wie schon erwähnt, bedeutet DBcc «Wenn die Bedingung cc nicht erfüllt ist, dann dekrementiere den Inhalt des Schleifenzählers Dn; wurde $FFFF noch nicht erreicht, dann weiter beim Label.» Die Codierung ist denkbar einfach:

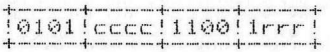


Bild 7.2 Die Codierung von DBcc

Für cccc muß der eben aufgelistete Bedingungsschlüssel eingesetzt werden und für rrr die Nummer des benutzten Datenregisters. Auf den Opcode folgt im nächsten Datenwort die 16-bit-Sprungweite, die genauso berechnet wird wie in 3.2 zur Adreßdistanz beschrieben.

8
Von bedingten und unbedingten Sprüngen

Eine Fortsetzung des Programmablaufs an anderer Stelle kann von einer Bedingung abhängig sein (bedingter Sprung) oder aber immer erfolgen (unbedingter Sprung). Die vorzeichenbehafteten Distanzen lassen sich in drei Gruppen einteilen:

Voller Adreßumfang (*32-bit-Distanz* entsprechend). Hier gibt es nur die unbedingten Sprungbefehle JMP (jump) (und JSR) mit absoluter Adressierung.

16-bit-Distanzen
Hier heißen die Befehle normalerweise «branch», bedingte Sprünge Bcc, unbedingte Sprünge BRA. Die Sprungweite folgt im nächsten Datenwort. JMP (und JSR) mit programmzählerrelativer Adressierung erlauben ebenfalls 16-bit-Distanzen.

8-bit-Distanzen
Hier bezeichnet man die Sprungweite als kurz (short), dementsprechend lautet die mnemonische Notierung Bcc.S bzw. BRA.S.

8.1 Die Codierung von BRA

BRA wird codiert 6000 dddd,
BRA.S wird codiert 60 dd.
Darin ist dddd bzw. dd die Adreßdistanz. Man erkennt, daß bei .S-Sprüngen nur ein Datenwort aus dem Speicher zu holen ist; das spart nicht nur Speicherplatz, sondern auch Zeit. Es ist nun nicht möglich (und auch nicht sinnvoll, kann aber beim Optimieren von Programmen durch Unachtsamkeit vorkommen und ruft dann verblüffende Effekte hervor), mit einem BRA.S

den nächstfolgenden Befehl zu erreichen. Die Adreßdistanz wäre nämlich null, denn der Programmzähler wird ja nach dem Holen des Opcodes sofort auf das nächste Datenwort gerichtet. Ein dd = 00 signalisiert aber dem Prozessor, daß kein .S-Sprung vorliegt, sondern ein 16-bit-Sprung, dessen Weite im nächsten Datenwort folgt. Noch einmal konkret: Die Opcodefolge 6000 4E75 würde nicht bedeuten «Mach einen Nullsprung und dann RTS»; der Prozessor interpretiert hier 4E75 als 16-bit-Sprungweite.

8.2 Die Codierung von Bcc

Wieder eine recht simple Codierung:

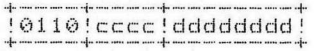


Bild 8.1 Die Codierung von Bcc

Für die Sprungweite dddddddd gilt unverändert das eben Beschriebene, und für die Codierung der Bedingung fast unverändert das in Abschnitt 7.4 Dargestellte – nur die beiden Bedingungsschlüssel T und F fehlen. Lassen Sie uns hierüber noch ein wenig nachdenken: Man könnte BRA als Bcc mit binärem Bedingungsschlüssel 0000 deuten, was T bedeuten würde: Dann wäre die Bedingung stets erfüllt, und das bedeutet «Sprung!», in Übereinstimmung mit der Bedeutung. Die Schreibweise BT gibt es jedoch nicht.

A.8.1: Der folgende Programmtorso reiht nur sinnlose Sprünge aneinander:

```
040000  xxxx          LABEL1: BRA.S   LABEL3
040002  xxxx xxxx     LABEL2: BHI     LABEL4
040006  xxxx          LABEL3: BHI.S   LABEL1
040008  xxxx xxxx     LABEL4: DBLT.S  D5,LABEL1
04000C  xxxx          LABEL5: BNE.S   LABEL5
04000E  xxxx xxxx     LABEL6: BRA     LABEL2
```

Bitte codieren Sie der Reihe nach Befehle und Sprungweiten.

8.3 Sortieren nach Merkmalen

```
040012
035C00                          ORG       $35C00
035C00
035C00  48E7 FFFE               MOVEM.L   D0-D7/A0-A6,-(A7)
035C04  41FA 003C               LEA       BUFER1(PC),A0
035C08  780F                    MOVEQ     #15,D4
035C0A  4298            LCLEAR: CLR.L     (A0)+
035C0C  51CC FFFC               DBRA      D4,LCLEAR
035C10  41FA 0020               LEA       BUFER0(PC),A0
035C14  43FA 002C               LEA       BUFER1(PC),A1
035C18  45FA 0038               LEA       BUFER2(PC),A2
035C1C  7A0F                    MOVEQ     #15,D5
035C1E  1018            LOOP:   MOVE.B    (A0)+,D0
035C20  6B0C                    BMI.S     NACHB1
035C22  14C0                    MOVE.B    D0,(A2)+
035C24  51CD FFF8       NEXT:   DBRA      D5,LOOP
035C28  4CDF 7FFF               MOVEM.L   (A7)+,D0-D7/A0-A6
035C2C  4E75                    RTS
035C2E
035C2E  12C0            NACHB1: MOVE.B    D0,(A1)+
035C30  60F2                    BRA.S     NEXT
035C32  12BF7887        BUFER0: DC.L      $12BF7887
035C36  0046FFAB                DC.L      $0046FFAB
035C3A  12AF0977                DC.L      $12AF0977
035C3E  FE987654                DC.L      $FE987654
035C42                  BUFER1: DS.B      16
035C52                  BUFER2: DS.W      8
035C62
035C62                          END
```

P.8.1 Sortieren nach Merkmalen

Unser Programm holt byteweise Daten, die mit der Assemblerdirektive DC (define constant) in Puffer 0 bereitgestellt wurden. Die negativen Zahlen sollen in Puffer 1 abgelegt werden, die positiven nach Puffer 2. Übersichtlichkeitshalber wird anfangs der Arbeitsbereich des Programms ab BUFER1 mit CLR (clear) auf Null gesetzt.

9
Die Bedienung des Rolf-D.-Klein-Assemblers

Rolf-Dieter KLEIN hat mit Konzept und Konstruktion seines NDR-Computers einen Volltreffer gelandet und eine Maschine herausgebracht, die in doppelter Weise die Potenz des M68000 (bzw. M68008 oder gar M68020) an den User weitergibt: einen akzeptablen und individuell ausbaubaren Speicherraum und ein Betriebssystem (hier Grundprogramm genannt), das in seinen erfreulicherweise nur 32 KB sowohl hochauflösende Screens unterstützt als auch einen residenten Assembler einschließt. Dieser soll die Reihe der dargestellten Assembler eröffnen.

9.1 Der Editor

Der Assembler übersetzt einen «Quelltext» in die Maschinensprache. Dieser muß so geschrieben sein, daß der Assembler ihn lesen kann, das heißt u. a., die einzelnen Elemente müssen nicht nur vereinbarten Normen entsprechen, sondern auch an vereinbarter Stelle im Speicher (bzw. auf Datenträger) zu finden sein. Decken Sie in P.8.1 die Zahlenkolonnen links ab, dann bleibt der Quelltext übrig.

Ein Quelltext wird in aller Regel mit einem Editor genannten Hilfsprogramm erstellt, wobei die TAB-Taste hilft, den Text in Kolonnen zu ordnen. Der Editor des NDR-Computers wird vom Menü aus aufgerufen und legt den eingetippten Text ab Default-Adresse $9000 im Speicher ab. Default ist ein Standard-Ersatzwert, der durch entsprechende Befehle wunschgemäß verändert werden kann. Wenn man mit einem Diskettenbetriebssystem arbeitet, so belegt dieses Speicherplatz und verlegt den Textanfang selbsttätig auf z. B. $10000.

9.2 Die ORG-Direktive

Direktiven (auch Pseudobefehle, sogar Pseudo-Ops genannt) sind Anweisungen, die nicht den Prozessor betreffen (also nicht zum Programm gehören), sondern die Arbeitsweise des Assemblers beeinflussen.

Nach der Programmüberschrift, die in Form eines (optionalen) Kommentars (eingeleitet durch «*») gesetzt wurde, folgt die ORG-Anweisung; auch sie muß nicht notwendigerweise verwendet werden. Ohne ORG legt der Assembler das Maschinenprogramm von einer Default-Adresse an im Speicher ab. Diese wird vom Grundprogramm auf $9C00 eingestellt, unter Systema-DOS aber auf $20000 verändert. (Wie schön, daß die 68000er-Programme normalerweise relokatibel sind!) In P.8.1 arbeitet der NDR dem Amiga in die Hand; durch Vorgabe von $35C00 als Startadresse gilt das erstellte Programm sogleich für einen anderen Computer. Man kann aber auch mit einigem Geschick verschiedene Programme nacheinander im Speicher durch Assemblieren erzeugen, sogar so gezielt, daß sie sich verketten lassen, obgleich hierfür andere Programmiertechniken bereitstehen.

(Eine Warnung an QL-Besitzer mit Metacomco-Assembler: Vor Einsatz der ORG-Direktive sollten Sie im Manual die diesbezügliche Anleitung sorgfältig lesen.)

Die Assemblerdirektive DC definiert eine im Speicher bereitzustellende Konstante.

9.3 Die eigentliche Assemblierung

Nach den Erklärungen von Abschnitt 5.5 nehmen wir an, daß der Quelltext für den ersten Assemblierungsversuch vorbereitet ist:

Aus dem Editor gelangt man mit der Tastenkombination CTRL-K-X zurück ins Menü und wählt dort (Menüpunkt «Optionen») die gewünschte Ausgabeweise aus, z. B. Ausgabe des kompletten Assemblerprotokolls auf Bildschirm oder Drucker, Ausgabe nur der fehlerhaften Zeilen auf Bildschirm oder Drucker.

Danach wird mit Option 2.2 «Assembler» die Assemblierung gestartet (ggf. nach Löschen der Symboltafel).

Freude bereiten die Fehlermeldungen, denn beim NDR-Computer ist die Korrektur höchst einfach. Man muß nicht – wie bei anderen zugegebenermaßen sehr leistungsfähigen Assemblern – erst das Editorprogramm laden und

dann den Quelltext, nach Korrektur wieder den Assembler laden und nochmals den Quelltext... Beim NDR-Computer sind alle diese Elemente zum sofortigen Zugriff bereit. Sogar wenn ein Probelauf des assemblierten Programms zum Systemabsturz führte, kann man getrost die Reset-Taste drücken; man findet auch dann noch den Quelltext unverändert vor und kann ihn korrigieren.

Sind nun alle Fehler behoben, kann das Programm sofort gestartet werden, denn beim Assemblieren wird der Objektcode schon von der mit ORG bezeichneten oder von der Default-Adresse an im Speicher abgelegt. (Man tut gut daran, ORG nicht in den Textspeicher oder in die Tabelle der Systemvariablen hineinzurichten!)

Ein guter Assembler liefert für die Programmdokumentation gute Unterlagen. Das Listing P.8.1 ist ein solches «Assemblerprotokoll».

9.4 Der Programmablauf

Mit der Option 1.2 «starten» wird der Programmstart eingeleitet. Der Computer fragt nach der Startadresse. Nach Eingabe von beispielsweise $35C00 <cr> wird der Ablauf gestartet.

Der NDR-Computer bietet mit Option 5.3 als ganz besonderen Komfort den Einzelschrittmodus. Nach Eingabe der Startadresse werden alle Registerinhalte, auch die der Kontrollregister, auf dem Bildschirm angezeigt, unten rechts neben dem Programmzählerstand auch der zur Abarbeitung anstehende Opcode. Noch schöner: Systema-DOS zeigt an dieser Stelle sogar die betreffende Zeile des Quelltextes mit Labels usw. an! Mit jedem cr (bzw. ENTER) geht es einen Schritt weiter. Man kann so wunderbar nicht zu lange Programmabläufe verfolgen, sich Feinheiten verdeutlichen oder aber ganz einfach Fehler suchen.

Wir werden uns nun nicht mehr mit dem Codieren aufhalten, sondern diese Aufgabe einem Assembler übertragen. Leser, die keinen Assembler besitzen, können die Opcodes den folgenden Beispielen entnehmen und werden nur im Ausnahmefall mit Hilfe eines Handbuchs den Objektcode ermitteln müssen.

10
Subroutinen und Stapel

Unterprogramme (Subroutinen) sind in sich abgeschlossene Programmteile, die nur einmal formuliert, aber wiederholt (mit BSR oder JSR) aufgerufen werden. Sie enden im Normalfall mit dem Befehl RTS (return from subroutine, Opcode 4E75), der dafür sorgt, daß der Programmablauf mit dem auf den Aufruf folgenden Befehl fortgesetzt wird.

10.1 Ein Beispiel

P.10.1 will die Struktur verschachtelter Subroutinen und die Sicherstellung des Rücksprungs aufzeigen. Das Hauptprogramm (Ebene 0) ruft das Unterprogramm der Ebene 1 auf, dieses das der Ebene 2 usw. Jede Ebene markiert ihre Spur im Puffer durch Hinterlassen ihrer Ordnungszahl im ASCII-Code.

Die Subroutinen werden hier mit BSR (branch to subroutine) aufgerufen, wobei für das Anhängsel .S auf 8.1 verwiesen wird. Wie aber findet der Prozessor z. B. nach dem Aufruf in 2001E bei Rückkehr aus der Subroutine zum nächsten (in 20020 stehenden) Befehl zurück? Vor der Beantwortung dieser Frage noch die kleine Randbemerkung, daß Assembler im allgemeinen als Eingaben nicht nur Hexadezimal- und Dezimalzahlen, sondern auch in Hochkommas eingeschlossene Strings akzeptieren und ihre Zeichen in ASCII-Bytes umwandeln.

10.2 Stapel und Stapelzeiger

Während der Bearbeitung eines BSR-Befehls legt der Prozessor die Adresse des folgenden Befehls in einem hierfür vorgesehenen Speicherbereich, dem Stapel (engl.: stack) ab. Diese Ablage erfolgt so, als ob sie durch den (in Wirklichkeit nicht existierenden) Befehl MOVE.L PC,–(A7) erzwungen würde.

```
020000  204F       EBENE0: MOVEA.L A7,A0
020002  4FFA 0056          LEA     STAPEL(PC),A7
020006  49FA 002E          LEA     BUFFER(PC),A4
02000A  18FC 0030          MOVE.B  #'0',(A4)+
02000E  6108               BSR.S   EBENE1
020010  18FC 0030          MOVE.B  #'0',(A4)+
020014  2E48               MOVEA.L A0,A7
020016  4E75               RTS
020018
020018  18FC 0031 EBENE1: MOVE.B  #'1',(A4)+
02001C  6106               BSR.S   EBENE2
02001E  18FC 0031          MOVE.B  #'1',(A4)+
020022  4E75               RTS
020024
020024  18FC 0032 EBENE2: MOVE.B  #'2',(A4)+
020028  6106               BSR.S   EBENE3
02002A  18FC 0032          MOVE.B  #'2',(A4)+
02002E  4E75               RTS
020030
020030  18FC 0033 EBENE3: MOVE.B  #'3',(A4)+
020034  4E75               RTS
020036
020036             BUFFER: DS.B    $24
02005A  4E71       STAPEL: NOP
```

P.10.1 Eine Subroutinenkaskade

A7 ist der Stapelzeiger (Stackpointer). Die Rückkehradresse wird mit Predekrement abgelegt, also nicht dort, wohin der Stackpointer zeigt, sondern darüber. Der Befehl NOP (no operation) am Ende des Programms ist an sich genauso sinnlos, wie es ein DS an dieser Stelle wäre, denn in 2005C wird ja gar nichts eingeschrieben, sondern «darüber». Nach Programmdurchlauf ist aber der Opcode 4E71 im Stapel ein willkommenes Orientierungsmerkmal.

Der Stapel wird als LIFO-Struktur oft mit einem Tellerstapel verglichen. Grundlage des Tellerstapels ist z. B. eine Tischplatte. Genauso, wie man ja in die Tischplatte keinen Teller hineinlegen kann, wird auch in die vom Stackpointer bezeichnete Adresse nichts eingeschrieben, sondern «darüber». Das ist beim Tellerstapel eindeutig; beim Stapel bedeutet es «zu fallenden Adressen hin».

LIFO-Struktur (last in first out) bedeutet, daß das als letztes eingeschriebene Element als erstes wieder herausgenommen (in Wahrheit nur kopiert) wird.

In unserem Beispiel kann man im Stapel die Rücksprungadressen 00020010, 00020020 und 0002002C und die Ablagerichtung («rückwärts») gut erkennen. Das wurde hier vereinfacht durch die Einrichtung eines Anwenderstapels, indem der zweite Befehl A7 auf die durch Label STAPEL bezeichnete Adresse richtete. Damit könnte aber der Prozessor am Programmschluß nicht wieder ins Betriebssystem zurückfinden, wenn nicht der ursprüngliche Stand von A7 wiederhergestellt würde. Wie ersichtlich, wurde der ursprüngliche Inhalt von A7 nach A0 zwischengespeichert.

10.3 Ein Subroutinennest

Klein, aber fein ist P.10.2. Die Grundstruktur wurde dem Betriebssystem des QL entnommen. (Sie bereitet dort Hexadezimalzahlen zu anzeigbaren Strings auf, vgl. P.14.4.) Hier wird mittels ADDQ in D2 gezählt, wie oft RTS erreicht wird, und jedes Erreichen im Puffer markiert. Was schätzen Sie, wie oft geschieht das?

Beim ersten RTS-Treff sehen Stapel und Stackpointer so aus: 020016 020012 02000E 02000A Rückad. (Die Position des Stackpointers wurde unterstrichen.) Das – verkürzt formuliert – 1. RTS führt nach 20016 zurück.

```
020000  47FA 001A          LEA      BUFER2(PC),A3
020004  4282               CLR.L    D2
020006  6100 0002          BSR      SURU1
02000A  6100 0002 SURU1:   BSR      SURU2
02000E  6100 0002 SURU2:   BSR      SURU3
020012  6100 0002 SURU3:   BSR      SURU4
020016  5202       SURU4:  ADDQ.B   #1,D2
020018  16C2               MOVE.B   D2,(A3)+
02001A  4E75               RTS
02001C             BUFER2: DS.B     100
020080
020080                     END
```

P.10.2 Ein Nest von Subroutinen

Vor dem 2. RTS wird kein BSR erreicht. Stapel also so: 020012 02000E 02000A Rückad. Zurück also nach 020012, wobei der Stapelzeiger schon auf 02000E weist. Jetzt wird ein BSR erreicht, 020016 auf dem Stapel abgelegt:

020016 02000E 02000A Rückad

Beim 3. RTS zurück nach 020016 und beim 4. nach 02000E mit (SP) = 02000A. Dabei wird der Stapel um zwei Einträge ab- und um drei aufgebaut:

020016 020012 02000E 02000A Rückad

Das 4. RTS führt nach 020016 zurück und damit auf

020012 02000E 02000A Rückad

und das 5. RTS auf

020016 02000E 02000 A Rückad

Es sei dem Leser überlassen, diese Spurenverfolgung bis zum Ziel fortzusetzen. Schneller bestätigt der umgekehrte Weg die Beobachtung:

Lautete der dritte Befehl nicht BSR SURU1, sondern BRA.S SURU4, so wurde das RTS nur einmal erreicht werden und ins Grundprogramm zurückführen.

Lautete der dritte Befehl aber BRA.S SURU3, würde der Programmschluß zunächst einmal erreicht werden und nach SURU4 zurückführen – insgesamt also zwei Durchläufe. So würde schließlich jede weitere Zurücknahme des Sprungziels die Anzahl der Durchläufe verdoppeln.

Abschließend sei der *BSR-RTS-Mechanismus* auf die beiden wesentlichen Punkte reduziert:

Beim Erreichen eines BSR
wird der Stapelzähler um vier vermindert und in dem hier beginnenden Langwort die Adresse des nächsten Befehls abgelegt.

Beim Erreichen eines RTS
wird der Inhalt des .L-Speicherbereichs, auf den der Stackpointer gerichtet ist, in den Programmzähler übernommen und der Stapelzeiger um vier erhöht.

M. a. W.: Der Stapelzeiger ist stets auf die Speicherstelle gerichtet, aus der das nächste Datum (bzw. Rücksprungadresse) zu holen ist.

A.10.1: Weshalb konnte in P.10.2 nicht der Befehl BSR.S eingesetzt werden?

11
Ein- und Ausgabe von Zeichen mit dem NDR-Computer

Ein weiterer Vorzug des NDR-Computers für das Erlernen der Maschinen-/Assemblersprache ist die Transparenz des Betriebssystems. Der Assembler erlaubt es sogar, Betriebssystemroutinen mit ihrem Namen, dem lediglich ein Masterspace («@», im Jargon auch «Klammeraffe» genannt) voranzustellen ist, anzusprechen. Der eigentliche Aufruf erfolgt mit dem Befehl JSR (jump to subroutine), der mit Adressierungsart «absolut lang» über den gesamten Adreßraum wirkt.

Während manche Assembler die Adressierungsart «absolut kurz» an der angegebenen Adresse erkennen, erwartet der RDK-Assembler einen Hinweis. Er übersetzt JSR $1234 mit 4EB9 0000 1234 und JSR $1234.S mit 4EB8 1234. Man kann so ganz nebenbei zwei Bytes einsparen, was beim üppigen Adreßbereich nicht sehr viel ausmacht. Wir wollen diese Möglichkeit vor allem nutzen, um im Interesse einer besseren Lesbarkeit unnötige Breiten der Listings zu vermeiden.

11.1 Die NDR-Subroutinen @CI und @CO2

@CI hält den Programmablauf an, bis über die Tastatur ein Zeichen eingegeben wird. Der ASCII-Code dieses Zeichens steht in D0.B. @CI ist eine reine Eingaberoutine ohne «Echo» auf dem Bildschirm.

@CO2 gibt das in D0.B im ASCII-Code stehende Zeichen normalerweise auf dem Bildschirm aus. Vorher muß der Grafikschirm mit @CLRSCREEN gelöscht werden.

Das Programm «Eingabe mit Echo» ist geradezu trivial. Da die CR/ENTER-Taste auch hier mit $0D codiert wird, braucht nur noch abgefragt zu werden, ob sie gedrückt wurde. In diesem Falle geht es über EXIT ins Betriebssystem zurück.

```
020000  4EB9 00003FEC            JSR     @CLRSCREEN
020006  4EB9 00000A00  LOOP:     JSR     @CI
02000C  0C00 000D                CMPI.B  #$0D,D0
020010  6708                    BEQ.S   EXIT
020012  4EB9 00000AE0            JSR     @CO2
020018  60EC                    BRA.S   LOOP
02001A  4E75              EXIT:  RTS
02001C                           END
```

P.11.1 Eingabe mit Echo

```
020000  4EB9 00003FEC            JSR     @CLRSCREEN
020006  41FA 001E                LEA     TEXT1(PC),A0
02000A  6104                    BSR.S   TXTAUS
02000C  41FA 002E                LEA     TEXT2(PC),A0
020010  1018              TXTAUS: MOVE.B  (A0)+,D0
020012  0C00 0000                CMPI.B  #0,D0
020016  6708                    BEQ.S   EXITT
020018  4EB9 00000AE0            JSR     @CO2
02001E  60F0                    BRA.S   TXTAUS
020020  4EF9 00001698  EXITT:   JMP     @CRLF
020026  436F6D70      TEXT1:   DC.L    $436F6D70
02002A  75746572                DC.L    $75746572
02002E  6E206D69                DC.L    $6E206D69
020032  74206465                DC.L    $74206465
020036  6D204E44                DC.L    $6D204E44
02003A  5200                    DC.W    $5200
02003C  69737420      TEXT2:   DC.L    $69737420
020040  696D6D65                DC.L    $696D6D65
020044  72206E6F                DC.L    $72206E6F
020048  63682076                DC.L    $63682076
02004C  69656C20                DC.L    $69656C20
020050  736368FC                DC.L    $736368FC
020054  6E657265                DC.L    $6E657265
020058  7200                    DC.W    $7200
02005A                           END
```

P.11.2 Für Leute mit Sinn für Unsinn

11.2 Die Ausgabe von Strings

Nach dem Initiieren des Bildschirms wird A0 auf einen String gerichtet, der mit dem vereinbarten Schlußbyte 00 endet, danach Ausgabe mit Subroutine TXTAUS. Anschließend wird A0 auf den nächsten String gerichtet, und das Programm fällt sozusagen von allein in die Subroutine TXTAUS hinein.

TXTAUS holt zunächst ein Zeichen aus dem Speicher nach D0. Wurde nicht das Schlußzeichen 00 geholt, dann erfolgt die Ausgabe mittels @CO2, und es geht zurück nach TXTAUS. Wurde aber das Schlußzeichen 00 geholt, dann geht es weiter bei EXITT: Die Subroutine @CRLF sorgt für Carriage Return und Linefeed. (Normalerweise wäre hier JSR @CRLF und RTS zu setzen. Da aber @CRLF als Subroutine mit RTS endet, kann der Aufruf mit JMP das Programm um eine Zeile verkürzen.)

Es war unnötig, die Subroutine TXTAUS in P.11.2 neu zu formulieren, denn sie ist als @WRITE bereits im Grundprogramm enthalten, jedoch – je nach Sichtweise – mit mehr Komfort und/oder mehr Aufwand. P.11.3 soll als Beispiel dienen.

Schrift wird bei allen dem Verfasser bekannten Computern mit dem M68000 als hochauflösende Grafik erzeugt. Der User hat damit nur wenig zu tun, denn die Feinausführung ist Sache des Grafikprozessors. Das erlaubt im vorliegenden Fall 256 verschiedene Schriftformen: 16 Breiten und 16 Höhen. Die Breite wird im MSN(D0) bereitgestellt, die Höhe im LSN; Beispiel: D0.B = $37 verursacht Schriftzeichen von der Breite 3 und der Höhe 7.

Der Bildschirm wird eingeteilt wie Quadrant I der Cartesischen Ebene: $0 \leq x < \$200$ und $0 \leq y < \$100$. x- und y-Koordinate der linken unteren Ecke der auszugebenden Schrift werden in D1.W und D2.W bereitgestellt. Diese Parameter wurden für acht verschiedene Ausgaben in der Tabelle PARA vorgegeben.

Nach dem schon bekannten @CLRSCREEN wird noch der hier unerwünschte Autoflip abgestellt: Beim NDR-Computer können vier Bildspeicher vollgeschrieben und in programmierbarer Folge ausgegeben werden. Die Routine @DELAY enthält eine quarzgesteuerte Warteschleife, die um so viele Zehntelsekunden verzögert wie in D0 angegeben, hier also um insgesamt eine Sekunde. Während dieser Zeit würde der Bildschirm blinken, was durch @SETFLIP mit Nullparametern unterbunden wurde.

Wie schon erwähnt, erspart der Aufruf einer Subroutine mit JMP (oder BRA) ein RTS, eine Programmzeile, zwei Bytes und eine halbe Mikrosekunde. Es fragt sich, ob der Verlust an Übersichtlichkeit erträglich ist.

```
020000  4EB9 00003FEC            JSR     @CLRSCREEN
020006  7000                    MOVEQ   #0,D0
020008  7200                    MOVEQ   #0,D1
02000A  4EB9 00001F06           JSR     @SETFLIP
020010  43FA 002A               LEA     PARA(PC),A1
020014  6100 0002               BSR     VIRFCH
020018  6100 0002       VIRFCH: BSR     DOPPEL
02001C  6100 0002       DOPPEL: BSR     TXTAUS
020020  41FA 0014       TXTAUS: LEA     TEXT(PC),A0
020024  4C99 0007               MOVEM.W (A1)+,D0-D2
020028  4EB9 00001386           JSR     @WRITE
02002E  700A                    MOVEQ   #10,D0
020030  4EF9 00000E5A           JMP     @DELAY
020036
020036  363830303000    TEXT:   DC.B    '68000',0
02003C  0002 0000 00C8  PARA:   DC.W    $02,000,200
020042  0017 01C2 0082          DC.W    $17,450,130
020048  0081 001E 0000          DC.W    $81,030,000
02004E  0044 0104 003A          DC.W    $44,260,058
020054  003F 01A4 0008          DC.W    $3F,420,008
02005A  00EC 0000 005E          DC.W    $EC,000,094
020060  0089 0000 000F          DC.W    $89,000,015
020066  0026 0140 0000          DC.W    $26,320,000
02006C
02006C                          END
```

P.11.3 Verschiedene Schriftgrößen

12
Stringausgabe mit dem QL

Das Programmbeispiel ähnelt P.11.2 und ist von mittlerem Aufwand. Vordergründig betrachtet wird der Usermodus des M68000 nicht verlassen (vgl. Kapitel 32).

12.1 Die Grobstruktur des Programmbeispiels

Platzenge zwang zur Verkürzung offizieller Namen: UTSCR (eigentlich UT_SCR (mit «Tiefstrich»)) ist eine nützliche (daher Utility) Routine, die ein Bildschirmfenster (SCReen) für die Ausgabe einrichtet. Hierzu gehört der Parameterblock SCRPA. Sein Aufbau ist .B .B .B .B .W .W .W .W, was im Quelltext zu drei Langwörtern zusammengezogen wurde. Im einzelnen bedeutet

05 hellblaue Farbe für den Fensterrahmen,
08 ist die Breite des Rahmens,
04 Farbe Grün für den Fenstergrund,
00 Farbe Schwarz für die Schrift;
00B0 Breite des Fensters (einschl. Rahmen),
0031 Höhe des Fensters (einschl. Rahmen),
009C x-Koordinate des linken Rahmenrandes,
0050 y-Koordinate des oberen Rahmenrandes.

Auf diesen Block ist A1 zu richten. Der eigentliche Aufruf wird im nächsten Abschnitt besprochen.

UT_SCR gibt bei erfolgreichem Aufruf in A0 eine Kanalkennzahl zurück, mit der bei evtl. späteren Ausgaben dieses Fenster (wieder über A0) angesprochen wird. Die beiden MOVEAs A0/A4 sind hier unnötig; sie sollen lediglich an diesen Sachverhalt erinnern und eine Möglichkeit aufzeigen, den

```
LOC     OBJECT     STMT        SOURCE STATEMENT

                    6
        =00C8       7 UTSCR EQU       $C8
        =00D0       8 UMTXT EQU       $D0
                    9
0000'   43FA 0026  10        LEA       SCRPA,A1
0004'   3478 00C8  11        MOVEA.W   UTSCR,A2
0008'   4E92       12        JSR       (A2)
000A'   2848       13        MOVEA.L   A0,A4
                   14
000C'   204C       15        MOVEA.L   A4,A0
000E'   3478 00D0  16        MOVEA.W   UMTXT,A2
0012'   43FA 0020  17        LEA       TEXT1,A1
0016'   4E92       18        JSR       (A2)
0018'   43FA 0026  19        LEA       TEXT2,A1
001C'   4E92       20        JSR       (A2)
001E'   43FA 002E  21        LEA       TEXT3,A1
0022'   4E92       22        JSR       (A2)
                   23
0024'   7000       24        MOVEQ     #0,D0
0026'   4E75       25        RTS
                   26
0028'   0508 0400  27 SCRPA  DC.L      $05080400
002C'   00B0 0031  28        DC.L      $00B00031
0030'   009C 0050  29        DC.L      $009C0050
0034'   0009       30 TEXT1  DC.W      9
0036'   2020 2020  31        DC.B      '    '
003A'   2051 4C20  32        DC.B      ' QL '
003E'   0A00       33        DC.W      $0A00
0040'   000B       34 TEXT2  DC.W      11
0042'   2020 6D69  35        DC.B      '  mi'
0046'   7420 5144  36        DC.B      't QD'
004A'   4F53       37        DC.B      'OS'
004C'   0A00       38        DC.W      $0A00
004E'   000C       39 TEXT3  DC.W      12
0050'   766F 6E20  40        DC.B      'von '
0054'   5369 6E63  41        DC.B      'Sinc'
0058'   6C61 6972  42        DC.B      'lair'
                   43
                   44        END
```

P.12.1 Stringausgabe mit dem QL

channel-ID (identifier) zwischenzuspeichern. (In P.10.1 wurde ganz ähnlich der Inhalt von A7 nach A0 «gerettet».)

Außerdem wird – was hier ignoriert wurde – in D0 ein Fehlercode zurückgegeben, wenn kein Fenster mit den angegebenen Parametern aufgemacht werden konnte. UT_MTEXT (im Quelltext UMTXT genannt) sendet eine Message (Botschaft) zu dem Kanal, dessen Kennummer (ID) in A0 steht. A1 muß auf den auszugebenden Text zeigen. Während dieser beim NDR-Computer mit einem Nullbyte enden mußte, ist hier die Stringlänge als Wort voranzustellen. Es ist wieder Sache des Programmierers, für eine gerade Anzahl von Zeichen zu sorgen. Soll eine neue Zeile folgen, endet der String mit einem Linefeed ($0A).

12.2 Vektorisierte Routinen (I)

Auch beim QL ist es möglich, einige Subroutinen mit ihrer absoluten Adresse aufzurufen, so z.B. UT_SCR mit JSR $3A6A und UT_MTEXT mit JSR $3A22. Was nun aber, wenn ein Kollege einen neueren QL mit einer neueren QDOS-Version besitzt? Es ist ja möglich, daß sich bei einer Fortschreibung des Betriebssystems gerade die kritischen Adressen änderten.

```
0000C0   2EC0 2F70 3A60 3A64 3A6A 3868 3872 3A00
0000D0   3A22 3A4A 3A50 0000 3016 3074 36FE 3714
0000E0   3742 3768 3792 3B0A 36D6 3796 40CE 4122
0000F0   3F66 3EC4 3F4C 3F48 3F44 3F20 3F1C 3F18

0000C0   302A 30DA 3A76 3A7A 3A80 39E2 39EC 3A14
0000D0   3A36 3A60 3A66 0000 3180 31DE 3878 388E
0000E0   38BC 38E2 390C 3B3C 3850 3910 404C 40AC
0000F0   BEE4 3E42 3ECA 3EC6 3EC2 3E9E 3E9A 3E96

0000C0   016A 016A 0016 0016 0016 017A 017A 0014
0000D0   0014 0016 0016 0000 016A 016A 017A 017A
0000E0   017A 017A 017A 0032 017A 017A FF7E FF8A
0000F0   FF7E FF7E FF7E FF7E FF7E FF7E FF7E FF7E
```

Bild 12.1 Auszug aus dem ROM des QL
oben: Version 1.03 (englischer QL)
Mitte: Version 1G13 (deutscher QL)
unten: Differenzen (1G13) – (1.03)

Bild 12.1 zeigt einen Auszug aus dem ROM des englischen und des deutschen QL. Man erkennt die unterschiedlichen Einsprungadressen in die einander entsprechenden Routinen. (An den Adreßdistanzen sind aber auch Gesetzmäßigkeiten in der Verschiebung ganzer Nester zu erkennen.) Für den Programmierer ändert sich dadurch nichts. Er ruft sowohl bei der deutschen als auch bei der englischen Ausführung des QL UT_SCR über den in 0000C8 eingeschriebenen .W-Vektor auf:

Der Besitzer eines *deutschen QL* lädt mit MOVEA.W $0000C8,A2 A2 mit $3A80, und JSR (A2) ist gleichbedeutend mit JSR $3A80.

Der Besitzer eines *englischen QL* lädt mit MOVEA.W $0000C8,A2 A2 mit $3A6A, und JSR (A2) ist gleichbedeutend mit JSR $3A6A.

12.3 Die Ermittlung der QDOS-Version

```
000' 7000                  MOVEQ    #0,D0
002' 4E41                  TRAP     #1
004' 41FA 0008             LEA      BUFFER,A0
008' 2082                  MOVE.L   D2,(A0)
00A' 7000                  MOVEQ    #0,D0
00C' 4E75                  RTS
00E' 0000 0000 BUFFER      DC.L     0
010' 0000 0000             DC.L     0
                           END
```

P.12.2 Die Ermittlung der QDOS-Version

P.12.2 ist nur als Mittel zum Zweck gedacht. Sein Ablauf kann eigentlich erst nach Kapitel 32 verstanden werden. Zusammen mit der Auswahl der Funktion 0 in D0 gibt der Manager-Trap MT_INF u.a. in D2.L die QDOS-Version als ASCII-String zurück. Sie wird hier im Puffer abgelegt und kann so z.B. leicht mit dem Monitorprogramm ermittelt werden. Die Ergebnisse sind:

beim dt. QL: D2 = 31473133 = '1G13' und
beim engl. QL: D2 = 312E3033 = '1.03'.

12.4 Auswege für den Amiga-User

In Kapitel 25 wird der K-Seka-Assembler für den Amiga besprochen. Das auf Seite 25 des Beihefts abgedruckte Beispielprogramm gibt 'Hello World' auf dem CLI-Fenster aus. Da weitere Information über den Amiga bis zur Stunde nicht erhältlich war, wird den Interessenten geraten, Ein- und Ausgaben vorerst mit einem BASIC-Hostprogramm (vgl. P.7.1H) zu realisieren.

13
Der Metacomco-Assembler (QL)

Der QL ist mit zwei Miniatur-Bandlaufwerken, Mikrodrives genannt, ausgestattet; sie heißen mdv1_ und mdv2_. Der Assembler wird mit einem brauchbaren Handbuch auf Cartridge (Mikrokassette) geliefert und ist für mdv1_ vorgesehen.

13.1 Eine Direktive und zwei Eigenarten

Natürlich stellt auch der Metacomco-Assembler mehrere Direktiven bereit. In P.12.1 wurde das bisher noch nicht besprochene EQU (equal) benutzt, das unter Assemblern weitverbreitet ist. Mit EQU wird hier beispielsweise dem Symbol UTSCR der Wert $C8 zugewiesen. Dadurch werden Programme leichter lesbar, denn man erkennt aus Zeile 11 sofort, daß in Zeile 12 die Subroutine UT_SCR aufgerufen wird.

Abweichend von den wohl meisten anderen Assemblern braucht hinter den Labels kein Doppelpunkt zu stehen; sie weisen sich nur durch ihre Positionierung in der ersten Kolonne des Quelltexts als solche aus. Diese Platzersparnis kann sehr angenehm sein, denn auf dem Bildschirm wird es manchmal ziemlich eng; deswegen auch hier die Verkürzungen, etwa von UT_MTEXT zu UMTXT.

Eine ganz wesentliche Eigenart dieses schönen Werkzeugs liegt in den beiden verschiedenen Arbeitsweisen – je nachdem, ob die ORG-Direktive verwendet wurde oder nicht:

Ohne ORG liefert der Metacomco-Assembler «position independent» (also relokatiblen) Objektcode; dementsprechend zeigt das Assemblerprotokoll auch keine absoluten Adressen, sondern relative, die zu einer im allgemeinen unbekannten Basisadresse zu addieren wären.

Wichtiger noch: Im «Ohne-ORG-Modus» erkennt der Metacomco-Assembler das «(PC)» nicht, sondern erwartet hinter «(» ein (Adreß-)Register:

Fehlermeldung! Wird die ORG-Direktive nicht verwendet, entfällt das Symbol der programmzählerrelativen Adressierung. Zum Vergleich zeigt Bild 13.1 ein entsprechendes Protokoll des Rolf-D.-Klein-Assemblers.

Bild 13.1 zeigt das aufs Wesentliche verkürzte Programm. LEA TEXT1,A1 lädt auch die Adresse von TEXT1 nach A1, doch mit Adressierungsart «absolut lang»; dementsprechend wird bei gleicher Wirkung (der Motorola-Norm entsprechend) anders codiert. Das Programm ist so aber nicht mehr relokatibel.

```
020000  43F9 00020028          LEA     SCRPA,A1
020006                         DS.W    6
020012  43F9 00020034          LEA     TEXT1,A1
020018  4E92                   JSR     (A2)
02001A  43F9 00020040          LEA     TEXT2,A1
020020  4E92                   JSR     (A2)
020022  43F9 0002004E          LEA     TEXT3,A1
020028                 SCRPA:  DS.W    6
020034                 TEXT1:  DS.W    6
020040                 TEXT2:  DS.W    7
02004E                 TEXT3:  DS.W    8
02005E
02005E                         END
```

Bild 13.1 Die unterschiedliche Interpretation des Quelltextes

13.2 Die Bedienung

Sie haben in mdv1_ eine Arbeitskopie des Assemblers und in mdv2_ ein zumindest formatiertes Band für die Aufnahme des Quelltextes. Mit exec mdv1_ed rufen Sie den Editor auf. Editor (und Assembler) sind Maschinenprogramme, und die werden mit exec geladen und gestartet. Sie sehen nun unten das niedrige Konsolfenster mit blinkendem Cursor, darüber das große Fenster mit der Meldung des Editors und ruhendem Cursor. Mit CTRL-C wird nun das Arbeitsfenster aktiviert, und Sie können die Frage «File name?» z.B. mit «mdv2_Pzwoelf_t» beantworten. Jetzt «weiß» der Editor, wo er einen evtl. schon in Arbeit befindlichen Quelltext finden kann und wie er heißt. Es wird empfohlen, Quelltexte durch Suffix _t zu kennzeichnen.

Die nächste Frage «Workspace size?» können Sie mit ENTER übergehen, wie auch die nun folgende nach einer evtl. Änderung des Fensterformats.

Ist der bezeichnete Quelltext tatsächlich am angegebenen Ort zu finden, wird er geladen und angezeigt. Anderenfalls erscheint im großen Arbeitsfenster ganz unten die Meldung «Creating new file», und oben links blinkt der Cursor, bereit zur Aufnahme des Quelltextes.

Nach der beabsichtigten Textbearbeitung oder -eingabe («END» nicht vergessen!) drücken Sie F3; mit dieser Taste wird der Editor zur Aufnahme von Kommandos aufgefordert: In der untersten Zeile des Arbeitsfensters erscheint als Bereitschaftszeichen ein Asterisk und daneben der blinkende Cursor. Jetzt wird der Quelltext z.B. mit «sa /mdv2_Pzwoelf_t/» (ENTER nicht vergessen) gesaved. Ist die Aufzeichnung abgeschlossen, wird die Kommandozeile nochmals aktiviert und «q» oder «Q» (quit) eingegeben. Das ist sehr wichtig, denn damit entfernt sich der Editor selbständig aus dem Speicher, und Sie haben genug Platz für den Assembler. Zunächst geht es aber mit CTRL-C zurück ins Konsolfenster.

Nach exec mdv1_asm meldet sich schließlich der Assembler. CTRL-C. Danach wird die Frage «Source file?» mit mdv2_Pzwoelf_t beantwortet. Der Assembler prüft nun, ob auf der angegebenen Peripherie der angeforderte Quelltext verfügbar ist. Wenn ja, wird neu gefragt: «Listing output (Y/N)?». Nehmen wir an, Sie haben es eilig und übergehen die Frage mit ENTER, die drei folgenden auch; dann wird nur verdeckt assembliert, wovon Sie eine ganze Menge hören, denn zunächst wird geprüft, ob ORG gesetzt wurde; je nachdem wird das erforderliche overlay (Zusatzprogramm) geladen. Dann die Meldung «Pass one: Reading source file». Der Quelltext wird gelesen und vorassembliert, wobei den Labels Zahlen zugeordnet werden (Aufstellung der Symboltafel). Meldung «Pass two: Reading source file»; der Quelltext wird noch einmal gelesen und unter Berücksichtigung der für die Symbole gefundenen Werte zum endgültigen Objektcode übersetzt («Zwei-Pass-Assembler»).

Ging alles gut, sehen Sie eine Meldung wie etwa: «No errors found in this Assembly. Code type: Position independent. Size 96 Bytes.»

Wenn Sie aber die Frage «Listing output?» mit Y beantworten, haben Sie im wesentlichen zwei Möglichkeiten, auf die folgende Frage «Listing file?» zu reagieren: «scr» führt zur Ausgabe des Protokolls auf dem Bildschirm, «ser1» (serielle Schnittstelle 1) zur Ausgabe auf dem Drucker. Nun sollten Sie «Code file?» mit mdv2_Pzwoelf_c beantworten, denn das Programm soll ja auch einmal ablaufen. Das Fenster braucht man dann nicht zu verändern. –

Der erzeugte Maschinencode wird nun auf den angegebenen Datenträger abgespeichert. Das «N», mit dem die Frage, ob noch weitere Quelltexte assembliert werden sollen, beantwortet wird, entfernt auch den Assembler aus dem Speicher. Das Maschinenprogramm kann nun mit exec mdv2_Pzwoelf_c geladen und gestartet werden. Das Suchen eines freien Platzes ggf. neben anderen und womöglich laufenden Programmen ist Sache des Betriebssystems.

Sie können aber auch in das Monitorprogramm die Zeile 125 LBYTES mdv2_Pzwoelf_c, anfad einfügen; sie lädt den Objektcode in den von RESPR reservierten Speicherraum. Das Maschinenprogramm kann nun mit dem H-Befehl betrachtet und mit dem G-Befehl gestartet werden.

13.3 Formales zum Metacomco-Assemblerprotokoll

Die Abkürzungen in der zweiten Zeile von Listing P.12.1 bedeuten:

LOC: Location, Speicherplatz, hier relativ zu der i.a. unbekannten Basisadresse (Anfangsadresse) des Programms.
OBJECT: Objektcode, Maschinenprogramm.
STMT: Hier Zeilennummer (der Anweisung).
SOURCE STATEMENT: Quellprogramm, die Folge der mnemonischen Befehle.

14
Umwandlung von Hexadezimalzahlen in Strings

Bekanntlich muß, um etwa die Zeichen «4» und «A» auf Bildschirm oder Drucker auszugeben, der Ausgabeeinheit der entsprechende ASCII-Code (hier $34 und $41) zugeführt werden. Für die sechzehn Hexadezimalziffern gilt folgende Zuordnung:

0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
30	31	32	33	34	35	36	37	38	39	41	42	43	44	45	46

14.1 Problemlösung in einzelnen Schritten

Im Eingabebereich (EBER) wird die umzuwandelnde Zahl bereitgestellt. Im Ausgabebereich (ABER) soll sie als ASCII-String 31 32 33 39 41 44 45 46 00 zur Ausgabe aufbereitet werden. Man darf dabei die systemspezifischen Vereinbarungen, beim NDR-Computer das abschließende Byte 00, nicht vergessen.

Zur Aufbereitung wird die hier achtstellige Hexadezimalzahl nach D3 geholt und insgesamt achtmal um vier Binärstellen nach links rotiert. Das höchstwertige Nibble wird dabei zum niederwertigsten; die anderen rutschen um je ein Nibble nach links. (Im folgenden Kapitel kommen wir auf diese wichtige Befehlsgruppe genauer zurück.) Nun wird das niederwertigste Byte von D3, bestehend aus der ehemals niederwertigsten Ziffer im MSN und der ehemals höchstwertigen im LSN (im Beispiel wurde 12 39 AD EF zu 23 9A DE F1), mittels Pointer A1 nach ABER übertragen. Dort wird das LSN von der Subroutine NIBASC zum ASCII-Byte aufbereitet, z.B. $xC zu $43.

NIBASC maskiert zuerst das MSN mittels ANDI (siehe Kapitel 16) weg, d.h., 3C (oder 7C usw.) wird zu 0C. Von den sechzehn verschiedenen Flagkombinationen, die der anschließende Vergleich mit $A theoretisch

```
020000  263A 0046        START:  MOVE.L   EBER(PC),D3
020004  43FA 0046                LEA      ABER(PC),A1
020008  7807                     MOVEQ    #7,D4
02000A  E99B             LOOP:   ROL.L    #4,D3
02000C  1283                     MOVE.B   D3,(A1)
02000E  6120                     BSR.S    NIBASC
020010  51CC FFF8                DBRA     D4,LOOP
020014  12BC 0000                MOVE.B   #0,(A1)
020018  4EB9 00003FEC    AUSGEB: JSR      @CLRSCREEN
02001E  41FA 0022                LEA      PARAM(PC),A0
020022  4C90 0007                MOVEM.W  (A0),D0-D2
020026  41FA 0024                LEA      ABER(PC),A0
02002A  4EF9 00001386    EXIT:   JMP      @WRITE
020030
020030  0211 000F        NIBASC: ANDI.B   #$0F,(A1)
020034  0C11 000A                CMPI.B   #$A,(A1)
020038  6502                     BCS.S    NOTADD
02003A  5E11                     ADDQ.B   #7,(A1)
02003C  0619 0030        NOTADD: ADDI.B   #$30,(A1)+
020040  4E75                     RTS
020042
020042  0021             PARAM:  DC.W     $0021
020044  0020                     DC.W     $0020
020046  0020                     DC.W     $0020
020048  1239ADEF         EBER:   DC.L     $1239ADEF
02004C                   ABER:   DS.W     5
020056                           END
```

P.14.1 Umwandlung einer Zahl in einen String

liefern könnte, interessieren nur zwei Möglichkeiten: a) Die Ziffer ist kleiner als $A; dann wird bei der internen Subtraktion Ziffer – $A Carry gesetzt – ein Zeichen dafür, daß nur $30 addiert werden muß, um den ASCII zu erhalten. b) Die Ziffer ist gleich oder größer $A; dann muß $37 addiert werden:

```
 0A        0B        ...  0F
+37       +37            +37
=41 ='A'= 42 ='B'...     =46 ='F'
```

Dies ist der Fall, wenn Carry gelöscht ist.

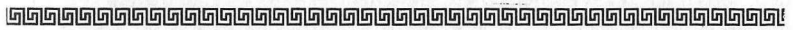
P.14.1 zerlegt diese Additionen in zwei Abschnitte: Wurde beim CMPI Carry gesetzt, braucht 7 nicht addiert zu werden, und es geht weiter bei NOTADD. Abschließend wird $30 mit Postinkrement addiert; das ergänzt den Zwischenstand zum fertigen ASCII und richtet A1 auf die nächste Ausgabeposition.

(Die Grundform des Additionsbefehls heißt ADD. Soll eine unmittelbar [immediate] gegebene Konstante addiert werden, wird ADD zu ADDI. Ist diese Konstante nicht größer als 8, kann sie innerhalb des Opcodes verschlüsselt werden; ADDI wird dann zu ADDQ.)

Nachdem dies alles achtmal geschehen ist, wird noch das Nullbyte angefügt. Bevor nun mit @WRITE ausgegeben werden kann, sind die Parameter Schriftbreite 2, Schrifthöhe 1, x-Position der Schriftausgabe und y-Position (ebenfalls 20) in die vereinbarten Register (vgl. P.11.3) zu laden, was hier mit dem schon mehrfach angewandten Befehl MOVEM geschieht, auf den nun auch bald näher einzugehen sein wird.

14.2 Fertige Problemlösung mit dem NDR-Computer

Das Grundprogramm bietet hierfür die Routinen @PRINT2X, @PRINT4X, @PRINT6X und @PRINT8X an. Die umzuwandelnde Zahl muß vor Aufruf in D0 stehen, und A0 ist auf den Ausgabebereich zu richten. Die @PRINT-Routinen wandeln die im LSB von D0 stehende zweistellige Hexadezimalzahl (bzw. die im LSW stehende vierstellige usw.) zum ASCII-String auf; sie hängen an den String auch noch das 00-Byte an, so daß anschließend bequem mit @WRITE ausgegeben werden kann.



```
020000  203A 0058          START:  MOVE.L  EBER(PC),D0
020004  41FA 0058                  LEA     ABER(PC),A0
020008  43FA 0038                  LEA     PARA(PC),A1
02000C  4EB9 00001668              JSR     @PRINT2X
020012  611A                       BSR.S   AUSGEB
020014  4EB9 00001660              JSR     @PRINT4X
02001A  6112                       BSR.S   AUSGEB
02001C  4EB9 00001658              JSR     @PRINT6X
020022  610A                       BSR.S   AUSGEB
020024  4EB9 0000164A              JSR     @PRINT8X
```

```
02002A  6002                    BRA.S   AUSGEB
02002C  4E75                    RTS
02002E
02002E  C147            AUSGEB: EXG     D0,D7
020030  4C99 0007              MOVEM.W (A1)+,D0-D2
020034  41FA 0028              LEA     ABER(PC),A0
020038  4EB9 00001386          JSR     @WRITE
02003E  CF40                    EXG     D7,D0
020040  4E75                    RTS
020042
020042  0041              PARA:  DC.W    #0041
020044  00200020                DC.L    #00200020

020048  0031                    DC.W    #0031
02004A  00200040                DC.L    #00200040
02004E  0021                    DC.W    #0021
020050  00200060                DC.L    #00200060
020054  0011                    DC.W    #0011
020056  00200080                DC.L    #00200080
02005A  1239ADEF        EBER:   DC.L    #1239ADEF
02005E                  ABER:   DS.W    5
020068                          END
```

P.14.2 Die Routinen @PRINTnX

In diesem Beispiel genügt es, D0 nur einmal mit der umzuwandelnden Zahl zu laden, da die Routinen @PRINTnX offensichtlich den Inhalt von D0 nicht verändern. Da die Ausgaben einander nicht überschreiben sollen, muß bei Verwendung von @WRITE jeweils ein Parameterblock übertragen werden, was hier mit Postinkrement geschieht, damit es gleich kontinuierlich weitergeht. In D0 muß aber hierbei die Schriftgröße übergeben und außerdem die umzuwandelnde Zahl aufbewahrt werden. Dieses Problem löst hier der Austausch der Registerinhalte mittels EXG (exchange). Zum Schluß sieht der Bildschirm etwa wie in Bild 14.1 aus.

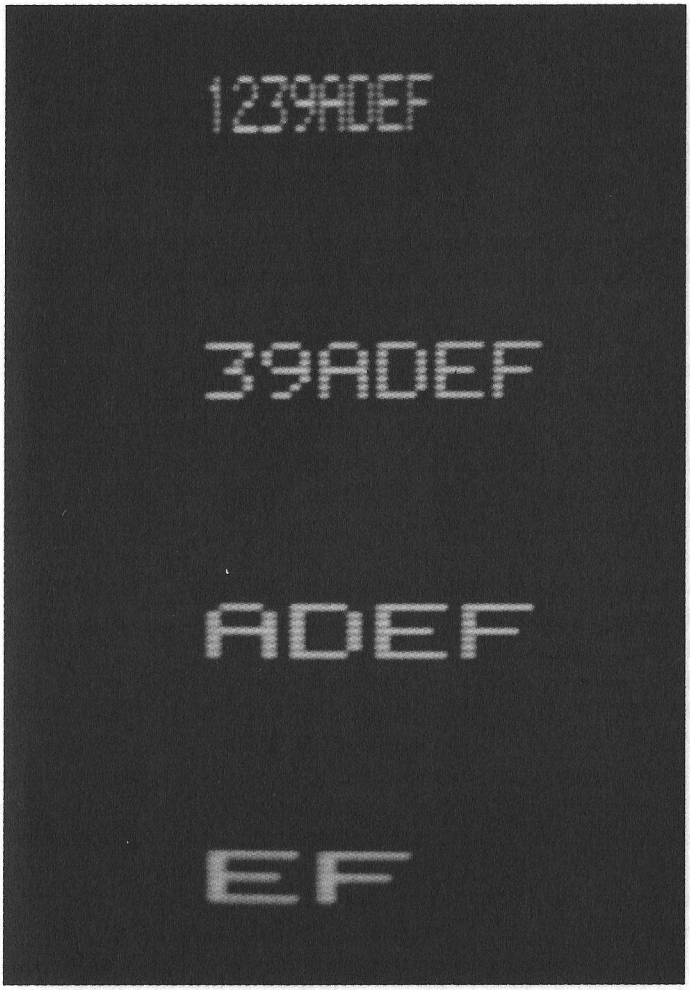


Bild 14.1 Das Ergebnis von Programm 14.2

14.3 Fertige Problemlösung mit dem QL

Den @PRINTnX-Routinen des NDR-Computers entsprechen ITOHB (Integer TO Hexadezimal Byte), ITOHW und ITOHL des QL. Da die zum String aufzubereitende Zahl hier im Speicher steht und über einen Pointer adressiert wird, werden ggf. nicht die niederwertigen, sondern die höherwertigen Stellen (MSB oder MSW) aufbereitet.

Zunächst wird wieder wie in P.12.1 ein Fenster aufgemacht, der channel-ID in A4 zwischengespeichert, und die Einsprungadresse zu UT_MTEXT in A3.

Beim Label BYTE beginnend werden Aufarbeitung und Ausgabe der ersten beiden Stellen der Beispielzahl 1239ADEF vorbereitet: Dem auszugebenden String muß die Angabe seiner Länge vorangestellt werden; diese beträgt hier – einschl. des angehängten LF – 3 Byte. Da sich das aber von Ausgabe zu Ausgabe ändert, müssen die Längen vom Hauptprogramm aus mitgeteilt werden. Nachdem A2 mit der Einsprungadresse der benötigten Umwandlungsroutine geladen wurde, geht es zur Konversion und Ausgabe zur Subroutine KNAUS.

Für die Routinen ITOHn geben DICKENS und OPIE (s. Anhang A) folgende Bereitstellungen an:

A1 ist auf die erste Stelle der umzuwandelnden Zahl zu richten, in unserer Bezeichnungsweise also EBER, von beiden genannten Autoren mit «stack» bezeichnet. A0 ist auf ABER zu richten dort «buffer» genannt. Wie im folgenden Abschnitt zu sehen sein wird, genügt das noch nicht; A6 muß auf Null gesetzt werden. Da aber A6 vom QDOS als Basisregister benutzt wird, muß es – hier auf den Stapel – gerettet und so schnell wie möglich wiederhergestellt werden. A6 wird durch Subtraktion A6–A6 auf Null gesetzt. Den Additionsbefehlen ADD, ADDA, ADDI, ADDQ, ADDX entsprechen die Subtraktionsbefehle SUB, SUBA, SUBI, SUBQ, SUBX.

Abschließend wird LF angehängt und ausgegeben. Dazu wird wie in P.12.1 UT_MTEXT benutzt, obgleich Sinclair diese Routine nicht empfiehlt und der begangene Weg nicht gegen alle Einwendungen gefeit ist.

P.14.3 Die Routinen ITOHB, ITOHW und ITOHl (QL) ▸

```
 LOC        OBJECT STMT          SOURCE STATEMENT
                     -
                     6
        =00FA        7 ITOHB EQU      $FA
        =00FC        8 ITOHW EQU      $FC
        =00FE        9 ITOHL EQU      $FE
        =00C8       10 UTSCR EQU      $C8
        =00D0       11 UMTXT EQU      $D0
                    12
 0000'  43FA 005A   13 START LEA      SCRPA,A1
 0004'  3478 00C8   14       MOVEA.W  UTSCR,A2
 0008'  4E92        15       JSR      (A2)
 000A'  2848        16       MOVEA.L  A0,A4
 000C'  3678 00D0   17       MOVEA.W  UMTXT,A3
                    18
 0010'  41FA 0056   19 BYTE  LEA      ABER,A0
 0014'  30FC 0003   20       MOVE.W   #3,(A0)+
 0018'  3478 00FA   21       MOVEA.W  ITOHB,A2
 001C'  6120        22       BSR.S    KNAUS
 001E'  41FA 0048   23 WORT  LEA      ABER,A0
 0022'  30FC 0005   24       MOVE.W   #5,(A0)+
 0026'  3478 00FC   25       MOVEA.W  ITOHW,A2
 002A'  6112        26       BSR.S    KNAUS
 002C'  41FA 003A   27 LANG  LEA      ABER,A0
 0030'  30FC 0009   28       MOVE.W   #9,(A0)+
 0034'  3478 00FE   29       MOVEA.W  ITOHL,A2
 0038'  6104        30       BSR.S    KNAUS
 003A'  7000        31 EXIT  MOVEQ    #0,D0
 003C'  4E75        32       RTS
                    33
 003E'  43FA 0018   34 KNAUS LEA      EBER,A1
 0042'  2F0E        35       MOVE.L   A6,-(A7)
 0044'  9DCE        36       SUBA.L   A6,A6
 0046'  4E92        37       JSR      (A2)
 0048'  2C5F        38       MOVEA.L  (A7)+,A6
 004A'  10BC 000A   39       MOVE.B   #$0A,(A0)
 004E'  204C        40 AUSGB MOVEA.L  A4,A0
 0050'  43FA 0016   41       LEA      ABER,A1
 0054'  4E93        42       JSR      (A3)
 0056'  4E75        43       RTS
                    44
 0058'  1239 ADEF   45 EBER  DC.L     $1239ADEF
 005C'  0000 0400   46 SCRPA DC.L     $00000400
 0060'  00F0 0050   47       DC.L     $00F00050
 0064'  0040 0040   48       DC.L     $00400040
 0068' =000C        49 ABER  DS.B     12
                    50
                    51       END
```

14.4 Das Subroutinennest ITOHn (QL)

```
ITOHL  003F18 6100 0002   BSR     ITOHW
ITOHW  003F1C 6100 0002   BSR     ITOHB
ITOHB  003F20 1036 9800   MOVE.B  00(A6,A1.L),D0
       003F24 E808        LSR.B   #4,D0
       003F26 6108        BSR.S   NIBAS
       003F28 700F        MOVEQ   #$0F,D0
       003F2A C036 9800   AND.B   00(A6,A1.L),D0
       003F2E 5249        ADDQ.W  #1,A1
NIBAS  003F30 0600 0030   ADDI.B  #$30,D0
       003F34 0C00 0039   CMPI.B  #$39,D0
       003F38 6302        BLS.S   NOTAD
       003F3A 5E00        ADDQ.B  #7,D0
NOTAD  003F3C 1D80 8800   MOVE.B  D0,00(A6,A0.L)
       003F40 5248        ADDQ.W  #1,A0
       003F42 4E75        RTS
```

P.14.4 Das Subroutinennest ITOHn (QDOS 1.03); kommentierter Disassemblerauszug

Es wird nicht schwerfallen, in P.14.4 die schon in P.10.2 und P.11.3 vorgestellte Vernestung von Subroutinen und den eigentlichen Konvertierungsalgorithmus von P.14.1 – wenn auch anders ausgeführt – wiederzuerkennen (ROL = Rotieren nach links; s. Abschnitt 15.2).

Wichtiger als das ist hier die Feststellung, daß innerhalb der Routinen A6 dreimal als Basisregister (s. Glossar) fungiert. Wenn aber in P.14.3 A0 auf ABER und A1 auf EBER gerichtet wurden, weisen – solange A6 einen von null verschiedenen Inhalt hat (was normalerweise der Fall ist) – die Summen A6 + A0 bzw. A6 + A1 in die Irre. Deshalb wurde A6 zwischendurch neutralisiert. (Die QDOS-adäquatere Lösung, A0 und A1 relativ zu A6 zu laden, wäre umständlicher gewesen.)

15
Rotieren und Shiften

Die hier besprochenen Befehle (Bild 15.1) verschieben Bitmuster um eine oder mehrere Binärstellen nach links oder rechts. Gelangt dabei das höchstwertige Bit in die niederwertigste Position (oder umgekehrt), heißt der Vorgang Rotieren (rotate), anderenfalls Shiften.

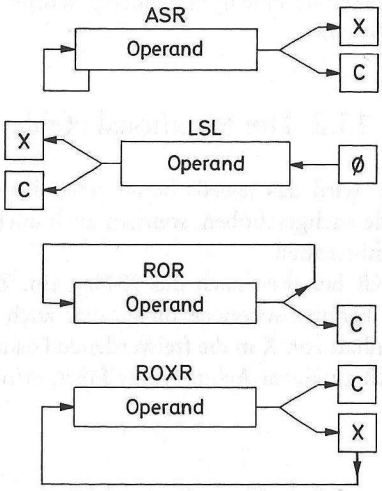


Bild 15.1 Einige Schiebe- und Rotationsbefehle

15.1 Die Shift-Befehle

Die beiden Befehle LSL und LSR verschieben das adressierte Datum um eine gewisse Anzahl von Binärstellen. Das jeweils sozusagen aus dem Datum herausgeschobene Bit gelangt sowohl ins Carry als auch ins Extend-Bit. Der freigewordene Platz wird mit einer Null aufgefüllt. Dieser Reigen heißt «logical shift left/right».

Während das arithmetische Links-Shiften ASL dem LSL gleicht, ist beim ASR ein wesentlicher Unterschied zu beobachten: Bei der arithmetischen Rechtsverschiebung bleibt das Vorzeichen erhalten:

Aus 0010 0111 wird 0001 0011, und
aus 1101 0111 wird 1110 1011:

War die Zahl negativ, war also ihr höchstes Bit gesetzt, wird eine 1 nachgeschoben, anderenfalls eine 0; mit anderen Worten: Das höchstwertige Bit reproduziert sich selbst.

15.2 Die Rotationsbefehle

Bei ROR und ROL wird das jeweils herausgeschobene Bit nicht nur am freigewordenen Ende nachgeschoben, sondern auch noch ins Carry kopiert; das X-Flag bleibt unbetroffen.

ROXL und ROXR beziehen auch das X-Flag ein. Zum eben Gesagten kommt hinzu: Das herausgeschobene Bit gelangt auch nach X; außerdem wird der bisherige Inhalt von X in die freiwerdende Position nachgeschoben: Während ein ROR.B zu einem Achterzyklus führt, erfolgt ein ROXR.B im Neunertakt.

15.3 Gemeinsamkeiten

Für jeden der acht Befehle bestehen drei verschiedene Syntaxen:

a) Der Operand steht im Speicher,
dann ist nur das .W-Format zugelassen, und es wird nur um eine Binärstelle geshiftet oder rotiert.
Syntax z.B. ASL <ea>

b) Der Operand steht in einem Datenregister:

Die Anzahl der Verschiebungen wird in drei Bits des Opcodes verschlüsselt; dabei bedeutet 000 die Höchstzahl von 8. Alle drei Formate sind zugelassen.

Syntax z.B. ROR.L #Anzahl,Dn

Die Anzahl der Verschiebungen ist in einem Datenregister verzeichnet; dabei sind alle drei Formate zugelassen.

Syntax z.B. LSL.B Dx,Dy

A.15.1: Berechnen Sie $\sum_{n=1}^{31} 2^n - 1$

Anleitung: Notieren Sie die Summanden binär.

16
Die logischen Verknüpfungen

Hier wird Bit mit Bit verknüpft, wofür es zunächst einmal drei Möglichkeiten gibt:

0011	0011	0011
AND 0101	OR 0101	EOR 0101
= 0001	= 0111	= 0110

16.1 AND

Ein Bit im Zieloperanden wird genau dann gesetzt, wenn beide einander entsprechenden Bits der Operanden (das eine UND das andere) gesetzt waren. Man findet hierfür gelegentlich Ausdrücke wie Verunden oder Undieren. Von Maskieren spricht man vor allem, wenn gezielt Teile eines Datums abgedeckt bzw. andere hervorgehoben werden sollen, wie z.B. in P.14.1, wo das MSN eines Bytes wegmaskiert wurde, um das LSN für sich allein aufarbeiten zu können. Ist die Maske unmittelbar (immediate) als Konstante gegeben, wird AND zu ANDI.

16.2 OR

Ein Bit des Zieloperanden wird gesetzt, wenn eines der entsprechenden Bits der Quelloperanden (oder beide) gesetzt waren. (Will man – etwa zur Konstruktion einer Schleife mit 64-bit-Zähler – feststellen, ob zwei Operanden gleichzeitig zu null wurden, kann man sie mit OR verknüpfen und das Ergebnis auf null prüfen.) Ist der eine Quelloperand unmittelbar gegeben, wird OR zu ORI.

16.3 EOR

Das Exklusiv-Oder setzt nur dann ein Bit des Ziels, wenn entweder das entsprechende Bit des einen Operanden oder aber das des anderen gesetzt war, nicht aber beide. Auch hierzu gibt es eine Immediate-Variante: EORI.

A.16.1: Wie kann man ermitteln, ob das Produkt zweier von null verschiedener Zahlen negativ ist?

16.4 NOT, NEG und NBCD

NOT und insbesondere NEG gehören nur bedingt zu den Verknüpfungen, denn man kann ja eigentlich nur zwei (oder mehr) Partner miteinander verknüpfen.

NOT (nicht) kann man sich noch als eine EOR-Verknüpfung mit −1 (bzw. FF usw.) vorstellen; hier wird das Komplement gebildet, indem jede 0 des Quelloperanden zu 1 und jede 1 zu 0 wird. Dabei ist stets Zahl + NOT(Zahl) = −1. Da hierbei die Summe zweier Bits mit gleichem Stellenwert immer 1 ist, spricht man auch vom Ein(s)erkomplement.

NEG negiert eine Zahl, kehrt ihr Vorzeichen um. Da hier Zahl + NEG(Zahl) = 0 sein soll, muß (vgl. NOT) das Ergebnis der Negation um 1 größer sein als das Komplement: NEG(Zahl) = NOT(Zahl) +1. Verkürzt läßt sich sagen: Wenn NOT das Einerkomplement bildet, erzeugt NEG wegen +1 (und folgenden Überträgen) das Zweierkomplement. NEGX bezieht in die Negation das Extendbit mit ein.

NBCD negiert ganz entsprechend binär codierte Dezimalzahlen, wobei ebenfalls das Extendbit einbezogen wird.

A.16.2: Gibt es Ausnahmen von der Regel, daß NEG das Vorzeichen einer Zahl umkehrt?

17
Die MOVEM-Befehle

Die MOVE-Multiple-Befehle transferieren mehrere Registerinhalte mit einem meistens nur zwei Wörter langen Befehl. Wir haben ihn bereits mehrfach verwendet, um mehrere Daten auf einmal zu übertragen (etwa die Parameterliste für @WRITE) oder sie vorübergehend zu sichern (vorsichtige Sicherung der Rücksprungparameter beim Amiga). Da mit den MOVEM-Befehlen auch die Inhalte der Adreßregister übertragen werden (können) und Adressen nur vier- oder achtstellig sein können, sind nur die .W- und die .L-Übertragung zugelassen.

17.1 Die Registerliste

Die Angabe, welche Registerinhalte im Speicher abgelegt oder von dort zurückgeholt werden sollen, wird mit hohem Komfort in der Registerliste angegeben: Die beiden Schreibweisen

D0–D1/D3/A4–A6 und D0/D1/D3/A4–A6

besagen dasselbe. Der Bindestrich kann als «bis» gelesen werden (A4 bis A6); der Schrägstrich trennt unzusammenhängende Angaben. Es ist gleichgültig, ob die Register «D0 bis D1» oder «D0 und D1» bewegt werden sollen.

Mit dieser Erklärung läßt sich die Syntax so angeben:

MOVEM.X Registerliste, <ea>
MOVEM.X <ea>,Registerliste

Erlaubt sind folgende Adressierungsarten
a) bei Ablage der Registerinhalte im Speicher:

(An), –(An), d(An), d(An,Rx), $xxxx und $xxxxxxxx;

b) beim Zurückholen aus dem Speicher:

(An), (An)+, d(An), d(An,Rx), $xxxx, $xxxxxxxx, d(PC) und d(PC,Rx).

17.2 Die Anordnung der Registerinhalte im Speicher

Wir benutzen noch einmal den Vergleich mit dem Tellerstapel, wo dem obersten Teller die niedrigste Adresse zugeordnet ist. In diesem Sinne liegt bei allen Adressierungsarten «oben» der Inhalt des Datenregisters mit der niedrigsten Ordnungszahl, «unten» der des Adreßregisters mit der höchsten Ordnungszahl.

17.3 Die Codierung der Registerliste

Sechzehn Register kommen für den multiplen Transfer in Betracht. Ihnen wird im zweiten Wort des Maschinenbefehls je ein Bit zugeordnet. Dieses Bit ist gesetzt, wenn das betreffende Register bewegt werden soll, anderenfalls gelöscht. Man kann sich die Codierung der Registerliste wohl am einfachsten so merken:

Bei der Ablage mit Predekrement entspricht die Codierung optisch unmittelbar der Registerliste:

Bit	15	14	13	12	...	08	07	...	03	02	01	00
	D0	D1	D2	D3	...	D7	A0	...	A4	A5	A6	A7

Bei allen anderen Adressierungsarten ist zu spiegeln:

Bit	15	14	13	12	...	08	07	...	03	02	01	00
	A7	A6	A5	A4	...	A0	D7	...	D3	D2	D1	D0

Das praktische Codieren soll nach wie vor von einem Assembler erledigt werden. Diese Erklärung will aber zum einen das Geschehen veranschaulichen, zum anderen wird sie in Kapitel 29 bei der Entwicklung eines Disassemblers von Nutzen sein.

18
Umwandlung von einem Zahlensystem ins andere

Für die größte in 32 bit erfaßbare vorzeichenlose Zahl gilt $FFFFFFFF = 4294967295. Für die größte achtstellige Dezimalzahl gilt umgekehrt 99999999 = $5F5E0FF. Die beiden folgenden Konvertierungsprogramme sollen so ausgelegt werden, daß sie den umfassenderen ersten Fall zu lösen vermögen.

18.1 Umwandlung DEZ → HEX

Bekanntlich ist 4711 = 4 * 1000 + 7 * 100 + 1 * 10 + 1. Hexadezimal notiert sähe das im Hornerschen Schema so aus:

$$\text{hex}(4711) = (((0 * A + 4) * A + 7) * A + 1) * A + 1$$

Zur Umwandlung einer vierstelligen Dezimalzahl ist also viermal «Zwischenergebnis mal zehn plus nächste Ziffer» auszuführen, wobei das erste Zwischenergebnis 0 ist.

Die beiden ersten Befehle in P.18.1 stellen die zum Einlesen eines Textes von der Tastatur erforderlichen Parameter bereit; außerdem ist nun A0 auf BUFFER gerichtet. Nach Eingabe von beispielsweise 123 mittels @READ (s. 20.2) sieht der Puffer so aus:

BUFFER: 31 32 33 00 ?? ?? ?? ?? ??,

und A0 zeigt auf das automatisch angehängte Byte 00.

Wir nehmen diese Vorgabe dankbar an, denn 1. erlaubt sie eine byteweise Auswertung, die zügiger bewerkstelligt werden kann als eine nibbleweise; 2. bietet sich das abschließende Nullbyte als Schlußzeichen für die Auswertung an.

```
020000  41FA 005A          START:  LEA       READPA(PC),A0
020004  4C98 000F                  MOVEM.W   (A0)+,D0-D3
020008  4EB9 00001486              JSR       @READ
02000E  41FA 0054                  LEA       BUFFER(PC),A0
020012  7000                      MOVEQ     #0,D0
020014  7200                      MOVEQ     #0,D1
020016  7400                      MOVEQ     #0,D2
020018  1418               LOOP:  MOVE.B    (A0)+,D2
02001A  671E                      BEQ.S     FERTIG
02001C  6102                      BSR.S     MALZEN
02001E  60F8                      BRA.S     LOOP
020020
020020  C0FC 000A          MALZEN: MULU      #10,D0
020024  C2FC 000A                  MULU      #10,D1
020028  0282 0000000F              ANDI.L    #$0F,D2
02002E  D282                      ADD.L     D2,D1
020030  4841                      SWAP      D1
020032  D041                      ADD.W     D1,D0
020034  4241                      CLR.W     D1
020036  4841                      SWAP      D1
020038  4E75                      RTS
02003A
02003A  4840               FERTIG: SWAP      D0
02003C  D041                      ADD.W     D1,D0
02003E  41FA 0030                 LEA       RESULT(PC),A0
020042  4EB9 0000164A             JSR       @PRINT8X
020048  41FA 0026                 LEA       RESULT(PC),A0
02004C  43FA 002E                 LEA       WRITPA(PC),A1
020050  4C91 0007                 MOVEM.W   (A1),D0-D2
020054  4EB9 00001386             JSR       @WRITE
02005A  60A4                      BRA.S     START
02005C
02005C  00110002           READPA: DC.L     $00110002
020060  00002000A                 DC.L      $00002000A
020064                     BUFFER: DS.B      12
020070                     RESULT: DS.B      12
02007C  00110080           WRITPA: DC.L      $00110080
020080  0002                      DC.W      $0002
020082                            END
```

P.18.1 Umwandlung DEZ → HEX

Die eigentliche Umrechnung erfolgt in D0, D1 und D2, die daher zuerst auf Null zu setzen sind. In D0 (höherwertige vier Hexadezimalstellen) und D1 (niederwertige vier Stellen) wird das Ergebnis aufgebaut; die ASCII-Bytes werden nacheinander mit Pointer A0 nach D2 geholt. Da schon beim bloßen MOVE-Befehl Flags beeinflußt werden, kann hier ohne CMPI das Schlußbyte 00 erkannt und die Aufbereitung abgebrochen werden.

Die Teilaufgabe «mal zehn plus Ziffer» wird in MALZEN ausgeführt. Der Befehl MULU multipliziert zwei vorzeichenlose (unsigned) .W-Zahlen ganz der Quelle-Ziel-Terminologie (vgl. Abschnitt 5.6) entsprechend zu einem .L-Produkt. Wird nun nach «mal zehn plus Ziffer» ein Zwischenergebnis länger als vier Stellen, so würde der Überhang nicht erfaßt werden und ginge verloren. Das verhindert die Aufteilung in Einer bis Tausender in D1 und Zehntausender bis Zehnmillionener in D0. Beide werden je für sich mit 10 multipliziert. Dann wird das Zonennibble des ASCII wegmaskiert und der signifikante Anteil zu D1.L addiert. So kann ein evtl. Übertrag noch die höherwertigen Stellen von D1 erreichen, die anschließend zu D0 addiert werden. Da sie aber zunächst nicht stellengleich stehen, werden sie aus dem MSW ins LSW geSWAPped. (Beim SWAP werden MSW und LSW eines Datenregisters vertauscht.) Anschließend muß der Überhang getilgt werden.

Ist das Schlußbyte erreicht, geht es weiter bei FERTIG: Die LSWs von D0 und D1 werden zusammengefügt, wozu wieder einmal zu swappen ist. @PRINT8X bereitet nun – vgl. P.14.2 – das in D0 stehende Ergebnis zum String auf, der in RESULT abgelegt und von dort mit @WRITE ausgegeben wird. Die hierzu erforderlichen Parameter werden in WRITPA bereitgestellt.

Das Programm mündet in eine nur gewaltsam mit Reset unterbrechbare Schleife, was in der großen EDV ein äußerst schlimmer Fehler wäre; hier aber soll ja nur an einem Personalcomputer mit geringstem Aufwand ein Algorithmus vorgestellt werden. Außerdem kontrolliert das Programm die Eingaben nicht auf Einhaltung der Vereinbarung und wandelt ungeniert DUENGER in 0045A710 um.

A.18.1:

a) Wieso?

b) Welche deutsche Stadt könnte sich hinter 1BCC8 verbergen?

Schließlich sei mitgeteilt, daß man das Ergebnis viel bequemer mit der Systemroutine @WERT hätte herleiten können. Diese hätte sogar zusammengesetzte Ausdrücke auswerten können!

18.2 Umwandlung HEX → DEZ

Nach HORNER ist

$$\$3BAD = (((0 * 16 + 3) * 16 + 11) * 16 + 10) * 16 + 13 = 15277.$$

Drei Teilaufgaben sind hierbei zu lösen:

1. Die Versechzehnfachung. Da die weitgesteckte Grenze auch mehr als achtstellige Zwischenergebnisse erbringt, kann nicht einfach in einem oder zwei Datenregistern, in denen leicht multipliziert werden könnte, aufgebaut werden. Wir verlegen die Multiplikation in den Speicher und kombinieren dort Shiften und Dezimaladdition.
2. Die einzelnen Hexadezimalziffern müssen vor der Addition dezimal aufbereitet werden, D zu 13 usw. Das könnte wieder durch Addition bzw. Subtraktion von 7 – je nach Ausfall eines Größenvergleichs – geschehen; abwechslungshalber wird hier eine Tabelle benutzt, was zugleich eine bequeme Prüfung auf Zulässigkeit der Eingabe erlaubt.
3. Die umgewandelten Hexadezimalziffern müssen ebenfalls dezimal addiert werden, wozu sich der Befehl ABCD (*a*ddiere *b*inär *c*odierte *D*ezimalzahlen) anbietet.

A.18.2: Schreiben Sie ein Programm, das die Summe 1234567823456789 + 5555555555555555 einmal dezimal und einmal hexadezimal berechnet.

P.18.2 geriet zugegebenermaßen ziemlich lang. Um dennoch die Übersicht zu behalten, wurde es in eine Vielzahl von Subroutinen zergliedert. Prüfen Sie selbst, ob dieser modulare Aufbau die Lesbarkeit verbessert.

18.2.1 Hauptprogramm und Eingabe

Das Hauptprogramm ruft nacheinander die Subroutinen EINGEBen, UMWANDeln, AUFBEReiten und AUSGEBen auf. Anschließend führt es zurück nach START, bildet aber dennoch keine endlose Schleife; sobald eine Nicht-Hexadezimalziffer eingetippt wird, springt es ins Betriebssystem zurück.

Die Subroutine EINGEB wurde im Prinzip schon in P.18.1 verwendet. Nach erfolgreicher Eingabe steht in EBER ein auf 00 endender String.

```
020000  610C            START:  BSR.S   EINGEB
020002  6124                    BSR.S   UMWAND
020004  6100 00AE               BSR     AUFBER
020008  6100 00C2               BSR     AUSGEB
02000C  60F2                    BRA.S   START
02000E
02000E                  * ========================== *
02000E
02000E  41FA 0010       EINGEB: LEA   READPA(PC),A0
020012  4C98 000F              MOVEM.W (A0)+,D0-D3
020016  41FA 00CC              LEA   EBER(PC),A0
02001A  4EF9 00001486           JMP     @READ
020020  00110002        READPA: DC.L    $00110002
020024  00020008                DC.L    $00020008
020028
020028                  * ========================== *
020028
020028  6112            UMWAND: BSR.S   INITRF
02002A  611A                    BSR.S   INITR2
02002C  43FA 00B6               LEA   EBER(PC),A1
020030  1019            LOPUMW: MOVE.B  (A1)+,D0
020032  6706                    BEQ.S   EXITUM
020034  6116                    BSR.S   SUCHTA
020036  613A                    BSR.S   ANFUEG
020038  60F6                    BRA.S   LOPUMW
02003A  4E75            EXITUM: RTS
02003C
02003C  41FA 00B0       INITRF: LEA   REFELD(PC),A0
020040  4298           INIT22:  CLR.L   (A0)+
020042  4258                    CLR.W   (A0)+
020044  4E75                    RTS
020046
020046  41FA 00AC       INITR2: LEA   REFEL2(PC),A0
02004A  60F4                    BRA.S   INIT22
```

P.18.2 Umwandlung HEX → DEZ (I)

18.2.2 Die eigentliche Umwandlung

Die Umwandlung erfordert einiges Rechnen, u. a. ist der Inhalt eines REchenFELDes zu dem eines anderen zu addieren. Anfangs werden beide auf null gesetzt (INITRF bzw. INITR2). Aus EBER wird ein ASCII geholt; ist es das Nullbyte, dann RTS, anderenfalls wird eine Tabelle abgesucht und der Zahlenwert des Bytes ermittelt und an das Zwischenergebnis angefügt.

18.2.3 Absuchen einer Tabelle und Anfügen

Die Tabelle HEXZIF enthält die sechzehn ASCII-Codes, die Hexadezimalziffern repräsentieren. Sie werden der Reihe nach mit dem in D0 stehenden Byte verglichen, wobei die Suchläufe in D1 ab 15 herabgezählt werden, so daß D1 immer den Wert des gerade geprüften Bytes enthält. Bei Gleichheit wird die Suchschleife ohne Dekrementieren verlassen; anderenfalls neuer Versuch. Wenn der Inhalt des Schleifenzählers D1 bis auf $FFFF vermindert wurde, ist das untersuchte Byte nicht Code einer Hexadezimalziffer. Hierfür ist keine spezielle Fehlermeldung, sondern lediglich ein Rücksprung ins Grundprogramm vorgesehen. Dazu müssen zwei Unterprogrammebenen übersprungen, also zwei im Stapel abgelegte .L-Rücksprungadressen ignoriert werden: Der Stackpointer wird um 8 erhöht, so daß das folgende RTS wie gewünscht ins Grundprogramm zurückführt.

Das «Anfügen» einer Stelle setzt sich aus Versechzehnfachen und Addieren zusammen – beides in Dezimalarithmetik. Von SECHZN aus wird viermal KOPDOP aufgerufen, KOPieren und verDOPpeln. Sechs Bytes sind von REFELD nach REFEL2 zu kopieren, ein .L und ein .W. Die anschließende Addition REFELD = REFEL2 + REFELD geschieht in sechs Schritten mit ABCD und Predekrement. Ein dabei evtl. entstehender Übertrag wird im X-Flag registriert und bei der nächsten Teiladdition zugefügt. Wenn aber vor der ersten Addition zufälligerweise X-Flag gesetzt war, wird die erste Teilsumme um eines zu groß. Aus diesem Grunde muß anfangs X-Flag gelöscht werden. Wir tun wieder einmal zuviel und setzen mit MOVE #0,CCR alle Flags des CCR zurück. Ganz wie wir es in der Schule gelernt haben, wird mit der niederwertigen Stelle («rechts») begonnen und dann nach links fortgesetzt.

Beim eigentlichen Anfügen (genauer: Addieren) der nächsten Ziffer muß daran gedacht werden, daß es zu Problemen führen kann, wenn man eine Hexadezimalziffer (genauer: einstellige Hexadezimalzahl) dezimal addieren

```
02004C                          ***** ***** ***** ***** *****
02004C                          *
02004C
02004C
02004C  45FA 0014               SUCHTA: LEA   HEXZIF(PC),A2
020050  720F                           MOVEQ   #15,D1
020052  B01A                    LOPSUC: CMP.B   (A2)+,D0
020054  57C9 FFFC                       DBEQ    D1,LOPSUC
020058  6602                           BNE.S   EXITSU
02005A  4E75                           RTS
02005C  4FEF 0008               EXITSU: LEA   08(A7),A7
020060  4E75                           RTS
020062  46454443               HEXZIF: DC.B    'FEDC'
020066  42413938                       DC.B    'BA98'
02006A  37363534                       DC.B    '7654'
02006E  33323130                       DC.B    '3210'
020072
020072                          * ------------------------------------------ *
020072
020072  6114                    ANFUEG: BSR.S   SECHZN
020074  61D0                            BSR.S   INITR2
020076  45FA 0081                       LEA   REFEL2+5(PC),A2
02007A  0C01 0009                       CMPI.B  #9,D1
02007E  6304                           BLS.S   NOTADD
020080  0601 0006                       ADDI.B  #6,D1
020084  1481                    NOTADD: MOVE.B  D1,(A2)
020086  6016                            BRA.S   ADDIRN
020088
020088  6106                    SECHZN: BSR.S   KOPDOP
02008A  6104                            BSR.S   KOPDOP
02008C  6100 0002                       BSR     KOPDOP
020090  45FA 005C               KOPDOP: LEA   REFELD(PC),A2
020094  2552 0006                       MOVE.L  (A2),06(A2)
020098  356A 0004 000A                   MOVE.W  04(A2),10(A2)
02009E  47FA 0054               ADDIRN: LEA   REFELD+6(PC),A3
0200A2  45EB 0006                       LEA   06(A3),A2
0200A6  7405                            MOVEQ   #5,D2
0200A8  44FC 0000                       MOVE    #0,CCR
0200AC  C70A                    LOPADD: ABCD    -(A2),-(A3)
0200AE  51CA FFFC                       DBRA    D2,LOPADD
0200B2  4E75                            RTS
```

P.18.2 Umwandlung HEX → DEZ (II)

will (vgl. A.18.3); Ziffern > 9 müssen erkannt und durch Addition von 6 in zweistellige Dezimalzahlen umgewandelt werden, z.B. $A + 6 = $10. Die Hexadezimalzahl $10 wird von ABCD als dezimal gegebene 10 aufgefaßt.

Vor dem eigentlichen «Anfügen» der neuen Stelle muß REFEL2 gelöscht werden; dann wird das durch CMPI #9,D1 geprüfte und ggf. aufbereitete Byte in die Position ganz rechts kopiert. Die anschließende Addition muß wieder die ganze Breite der Rechenfelder erfassen.

A.18.3: Entwerfen Sie ein Programm, das ohne unnötigen Aufwand prüft, wie weit die Dezimaladdition von Hexadezimalzahlen richtig abläuft.

18.2.4 Stringaufbereiten und Ausgabe

Da eine achtstellige Hexadezimalzahl einer maximal nur zehnstelligen Dezimalzahl entspricht, ist die Aufbereitung überdimensioniert; es sei dem Leser überlassen, das zu ändern. Die Umwandlung des Ergebnisses in einen String übernehmen die Routinen PRINTnX, wobei es äußerst angenehm ist, daß sie das Nullbyte automatisch anfügen, den Pointer aber auf ihm stehen lassen, so daß sofort weiter umgewandelt werden kann.

Die abschließende Ausgabe bedarf keiner weiteren Erwähnung.

```
0200B4                          ***** ***** ***** ***** *****
0200B4                          *
0200B4
0200B4                          * ============================== *
0200B4
0200B4  43FA 0038       AUFBER: LEA   REFELD(PC),A1
0200B8  41FA 0040               LEA   ABER(PC),A0
0200BC  2019                    MOVE.L  (A1)+,D0
0200BE  4EB9 0000164A           JSR     @PRINT8X
0200C4  3019                    MOVE.W  (A1)+,D0
0200C6  4EF9 00001660           JMP     @PRINT4X
0200CC
0200CC                          * ============================== *
0200CC
0200CC  41FA 002C       AUSGEB: LEA   ABER(PC),A0
0200D0  43FA 000C               LEA   WRITPA(PC),A1
0200D4  4C91 0007               MOVEM.W (A1),D0-D2
0200D8  4EF9 00001386           JMP     @WRITE
```

```
0200DE
0200DE  00110080          WRITPA: DC.L    $00110080
0200E2  0002                      DC.W    $0002
0200E4
0200E4                    * ========================== *
0200E4
0200E4                    EBER:   DS.B    10
0200EE                    REFELD: DS.B    6
0200F4                    REFEL2: DS.B    6
0200FA                    ABER:   DS.B    14
020108
020108                            END
```

P.18.2 Umwandlung HEX → DEZ (III)

19
Das Sieb des Eratosthenes

ERATOSTHENES lebte etwa von 275 bis 214 v. Chr. im griechischen Mittelmeerraum. Er ermittelte nicht nur als erster (aufgrund eines bewunderungswürdigen Gedankengangs) den Erdumfang, sondern konstruierte ein Sieb, in das man oben «alle» Zahlen hineinschütten konnte. Es ließ die zusammengesetzten Zahlen durchfallen, und übrig blieben die Primzahlen.

Es gibt andere Algorithmen, doch entbehrt es nicht eines nostalgischen Reizes, einen Gedankengang aus der griechischen Antike mit der Technik des ausgehenden zwanzigsten Jahrhunderts nachzuvollziehen.

19.1 Problemanalyse

ERATOSTHENES schrieb «alle» natürlichen Zahlen der Reihe nach auf. Die erste dieser Zahlen, 1, strich er aus, weil ja jede Zahl durch 1 teilbar ist. Die als erste stehengebliebene Zahl 2 war die erste Primzahl; er soll sie eingekringelt haben. Hier beginnend strich er nun «jede» zweite Zahl aus, denn sie waren ja durch zwei teilbar, also keine Primzahlen. Eingekringelt wurde dann die

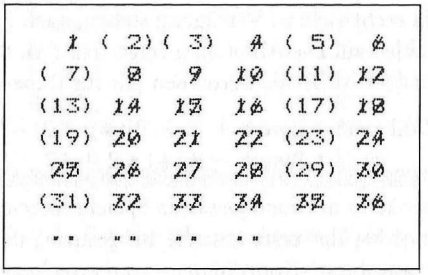


Bild 19.1 Das Sieb des Eratosthenes

Drei, und jede dritte Zahl durchgestrichen. Besonders übersichtlich wird das Bild, wenn man die Zahlen sechsspaltig anordnet (Bild 19.1).

Nimmt man die 2 aus oder stellt man sie voran, so kann man jede zweite Kolonne ungesehen ausstreichen; jede dritte auch, doch das würde die Transparenz des Programms mindern. Wir übertragen diese Verkürzung auf die Befehlsstruktur des M68000 (Bild 19.2).

Byte0	3	5	7	9	11	13	15	17
Bit#	7	6	5	4	3	2	1	0
Byte1	19	21	23	25	27	29	31	33
Bit#	7	6	5	4	3	2	1	0
Byte2	35	37	39	41	43	45	47	49
Bit#	7	6	5	4	3	2	1	0
Byte3	51	53	55	57	59	61	63	65
Bit#	7	6	5	4	3	2	1	0

Bild 19.2 Die Byte/Bit-Struktur des Siebes

Ausgehend von der Adressierungsart d(An,Dx.W) und unter Berücksichtigung der dekrementativen Schleifenstruktur, ordnen wir den Bits des Speichers, von denen ja recht viele zu Verfügung stehen, nach Maßgabe von Bild 19.2 natürliche Zahlen zu. Die Abbildung zeigt, daß z.B. dem Bit5 in Byte3 die Zahl *55* zugeordnet ist. Evidentermaßen gilt für diese Codierung:

$$\begin{aligned} \text{Zahl} &= (8 * \text{Byte\#} + (7 - \text{Bit\#})) * 2 + 3 \\ &= (8 * \text{Byte\#} - \text{Bit\#}) * 2 + 17 \end{aligned}$$

P.19.1 muß also zunächst in einem gewissen Speicherbereich alle Bits setzen. Dann wird von vorn her das erste gesetzte Bit gesucht; ihm entspricht eine Primzahl. Sie wird aus ihren «Koordinaten» zu n errechnet und ausgegeben. Danach wird jedes n-te Bit gelöscht, und das Spiel beginnt von vorn.

19.2 Das Eratosthenes-Programm für den NDR-Computer

```
020000
= 0000001E                  FIN     EQU     30
020000
020000  47FA 0084           INIMEM: LEA     ZAHLEN(PC),A3
020004  303C 001F                   MOVE.W  #FIN+1,D0
020008  16FC 00FF           LOPINI: MOVE.B  #$FF,(A3)+
02000C  51C8 FFFA                   DBRA    D0,LOPINI
020010
020010  47FA 0074           SUCHZL: LEA     ZAHLEN(PC),A3
020014  7800                       MOVEQ   #0,D4
020016  7A07               LOPAUS: MOVEQ   #7,D5
020018  0BB3 4000           LOPINN: BCLR    D5,0(A3,D4.W)
02001C  6612                       BNE.S   GEFUND
02001E  51CD FFF8                   DBRA    D5,LOPINN
020022  0644 0001                   ADD.W   #1,D4
020026  0C44 001E                   CMPI.W  #FIN,D4
02002A  6BEA                       BMI.S   LOPAUS
02002C  7000                       MOVEQ   #0,D0
02002E  4E75                       RTS
020030
020030  3004               GEFUND: MOVE.W  D4,D0
020032  E740                       ASL.W   #3,D0
020034  9045                       SUB.W   D5,D0
020036  D040                       ADD.W   D0,D0
020038  0640 0011                   ADDI.W  #17,D0
02003C  3C00                       MOVE.W  D0,D6
02003E
02003E  41FA 003C                   LEA     ABER(PC),A0
020042  4EB8 15FC                   JSR     @PRINT4D.S
020046  43FA 002E                   LEA     WRITPA(PC),A1
02004A  4C91 0007                   MOVEM.W (A1),D0-D2
02004E  41FA 002C                   LEA     ABER(PC),A0
020052  4EB8 1386                   JSR     @WRITE.S
```

P.19.1 Das Sieb des Eratosthenes (I) für NDR

Das Programm ist auch zum Experimentieren geeignet: Der Variablen FIN wird zunächst nur eine kleine Zahl zugewiesen; ist das Programm ausgetestet, kann man eine größere einsetzen, in wenigen Sekunden assemblieren, und in etwa einer halben Minute kann ein neuer Lauf gestartet werden. – Die ersten vier Zeilen initiieren den Speicher (memory), setzen im eingestellten Bereich alle Bits auf 1.

SUCHZL sucht die erste Primzahl. Der Bytezähler D4 wird auf Null gesetzt, und der Bitzähler D5 mit 7, der höchsten Ordnungszahl eines Bits im Byte, geladen. Der folgende Befehl bedarf einer Erklärung: BCLR löscht das adressierte Bit. Dieses steht hier im Speicher, weshalb nur Bits innerhalb eines Bytes angesprochen werden können. BCLR tut noch mehr: Je nachdem, ob das adressierte Bit gesetzt/gelöscht war, wird Zeroflag gelöscht/gesetzt. War das Bit gesetzt, so wurde eine (von ihm repräsentierte) Primzahl gefunden, und es geht bei GEFUND weiter.

War das geprüfte Bit gelöscht, wird in einer inneren Schleife (LOPINN) der Bitzähler D5 dekrementiert. Ist er ohne Erfolg heruntergezählt worden, wird der Bytezähler um 1 erhöht, und es geht zurück zu einer neuen äußeren Schleife, die mit Neuinitiierung des Bitzählers beginnt. Wenn aber der Bytezähler die vorgegebene Grenze FIN überschritten hat, geht es mit RTS zurück ins Betriebssystem. Das MOVEQ zuvor ist eine freundliche Erinnerung für die QL-Fans.

Wurde ein gesetztes Bit GEFUNDen, wird zunächst nach der oben mitgeteilten Formel die Primzahl aus ihren Koordinaten berechnet. PRINT4D wandelt nach den beinahe schon bekannten Regeln eine in D0 stehende vierstellige Hexadezimalzahl in eine Dezimalzahl um und legt sie als String in ABER ab. Die Systemroutine @WRITE ist bereits gut bekannt.

Nachdem eine Primzahl p erreicht und ausgegeben und das sie repräsentierende Bit gelöscht wurde, muß jedes p-te Bit gelöscht werden, denn es verkörpert ja Vielfache von p. Das geschieht fast ohne Mathematik ab LOPLOE (Löschschleife): Der Bitzähler D5 wird (der dekrementativen Struktur entsprechend) um p vermindert. Wird er dabei unterzählt, wird er so lange um 8 erhöht, bis er wieder nichtnegativ geworden ist; jedesmal wird der Bytezähler D4 um 1 erhöht. Nach dem Löschen der so erreichten Bits geht es im Minus-p-plus-8-Schritt weiter, bis D4 die gewählte Grenze FIN überschreitet.

Dann ein kleiner Komfort, ohne den man die schnell ausgegebenen Ergebnisse nicht mehr verfolgen könnte: Die Systemroutine @CSTS wartet auf eine Tastatureingabe; wird also keine Taste gedrückt, kann man das Ergeb-

```
020056  9A46                LOPLOE: SUB.W   D6,D5
020058  6A06                LOPLO2: BPL.S   NOTADD
02005A  5244                        ADDQ.W  #1,D4
02005C  5045                        ADDQ.W  #8,D5
02005E  60F8                        BRA.S   LOPLO2
020060  0BB3 4000          NOTADD: BCLR    D5,0(A3,D4.W)
020064  0C44 001F                  CMPI.W  #FIN+1,D4
020068  65EC                        BCS.S   LOPLOE
02006A
02006A  4EB8 09D8          WARTEN: JSR     @CSTS.S
02006E  67FA                        BEQ.S   WARTEN
020070  4EB8 0A00                  JSR     @CI.S
020074  609A                        BRA.S   SUCHZL
020076
020076  00110002            WRITPA: DC.L    $00110002
02007A  0002                        DC.W    $0002
02007C                      ABER:   DS.B    10
020086                      ZAHLEN: DS.B    FIN+1
0200A5
0200A5                              END
```

P.19.1 Das Sieb des Eratosthenes (II) für NDR

nis in Ruhe ansehen. Weiter geht es auf Tastendruck, dann aber in schneller Folge «ohne Ende»: Der Status «Taste gedrückt» bleibt so lange erhalten, bis von der Tastatur das auslösende Zeichen eingelesen wurde, was mit @CI geschieht. Das Zeichen könnte nun dazu benutzt werden, den Ablauf zu unterbrechen und ins Betriebssystem zurückzukehren. Hierauf wurde verzichtet. Erfreulich: Solange man eine Taste gedrückt läßt, läuft das Programm weiter. Man kann es nach Gutdünken und Augenschein unterbrechen, beispielsweise wenn man einen bestimmten Größenbereich nach Primzahlen absuchen will. Nach jeder Ausgabe geht es wieder zurück nach SUCHZahL.

19.3 Das Eratosthenes-Programm für den QL

Das Programm für den QL ist zum großen Teil eine fast wörtliche Abschrift des NDR-Programms. Drei Unterschiede: a) Die Labels wurden dem etwas «beengten» Bildschirm entsprechend verkürzt; sie werden aber einen Vergleich kaum erschweren. b) Wie unter 13.1 besprochen, fällt das «(PC)» weg. c) Die Ausgabe wurde in eine Subroutine verlegt.

AUSKI ist eine Zusammenziehung von AUSgabe und KeyIn. UT_MINT (hier MINT) gibt den Inhalt von D1.W als Dezimalzahl auf dem ausgewählten Kanal aus (entspricht also in etwa @PRINT4D und @WRITE zusammen); leider von $8000 an als negative Zahl. UT_MINT wird über den Vektor $CE erreicht. Wer das Fensterln auf dem Bildschirm nicht schätzt, benutzt das normale große Arbeitsfenster «Screen», das mit Kanalkennzahl $00010001 in A0 zu erreichen ist.

QDOS bietet eine Fülle von Screen drivers (Bildschirmkontrollroutinen) an, die mit einer Funktionsauswahl in D0 über den Befehl TRAP #3, der erst viel später besprochen werden soll, erreichbar sind. Mit SD_NL (hier SDNL) = $12 in D0 wird so eine neue Zeile (new line) erzeugt. Von fast allen vergleichbaren QDOS-Routinen wird in D0 eine Fehlermeldung zurückgegeben. Hier interessiert nur die Meldung −4 «Out of range», auf deutsch: Der Bildschirm ist voll, es muß gescrollt werden.

SD_SCROL (hier SCRL) = $18 ist die benötigte Funktionsnummer. −10 besagt, daß der Bildschirminhalt zwecks Einrichtung einer neuen Zeile um 10 Pixels nach oben geschoben werden soll (die zu verringernde y-Koordinate zählt vom oberen Rand ab). Noch eine QDOS-Spezialität: In D3 wird eine Zeitbegrenzung (timeout) vorgegeben, hier («1») die Dauer einer Wechselstromschwingung (1/50 s): Konnten die Scrollparameter schon vor Ablauf dieser Zeit an den Grafikprozessor übergeben werden, geht es weiter; anderenfalls wird das Scrollen ignoriert (wenn z.B. in einem Multiusersystem ein anderer Teilnehmer Vorrang hatte).

Nun würde es in nach rechts geneigten Treppen weitergehen, wenn man nicht bestimmte, daß das nächste Zeichen am Zeilenanfang ausgegeben werden soll. SD_TAB macht's möglich: D1 (= 0) bestimmt die Position, in D3 wieder das timeout.

KEYIN: Unter dem großen «Screen»-Fenster, durch das nur ausgegeben werden kann, liegt das vier Zeilen hohe «Console»-Fenster mit Kanalkennzahl $00000000. Über diesen Channel werden u.a. Tastatureingaben aufgenommen und ihr Echo ausgegeben. Mit D0 = 1 wird IO_FBYTE (fetch a

```
LOC          OBJECT  STMT                SOURCE STATEMENT
                       1 * P.19.2 ERATOSTHENES
                       2
      =0001            3 FBYTE EQU       1
      =01F4            4 FIN   EQU       500
      =00CE            5 MINT  EQU       $CE
      =0012            6 SDNL  EQU       $12
      =0018            7 SCRL  EQU       $18
      =0011            8 TAB   EQU       $11
      =00010001        9 SCHI  EQU       $00010001
      =0000           10 TAST  EQU       $00000000
                      11
0000'  47FA 0092      12 INIM  LEA       ZALEN,A3
0004'  303C 01F5      13       MOVE.W    #FIN+1,D0
0008'  16FC 00FF      14 LINI  MOVE.B    #-1,(A3)+
000C'  51C8 FFFA      15       DBRA      D0,LINI
                      16
0010'  47FA 0082      17 SUZL  LEA       ZALEN,A3
0014'  7800           18       MOVEQ     #0,D4
0016'  7A07           19 LPAU  MOVEQ     #7,D5
0018'  0BB3 4000      20 LPIN  BCLR      D5,0(A3,D4.W)
001C'  6612           21       BNE.S     GFUN
001E'  51CD FFF8      22       DBRA      D5,LPIN
0022'  D87C 0001      23       ADD.W     #1,D4
0026'  0C44 01F4      24       CMPI.W    #FIN,D4
002A'  6BEA           25       BMI.S     LPAU
002C'  7000           26       MOVEQ     #0,D0
002E'  4E75           27       RTS
                      28
0030'  3204           29 GFUN  MOVE.W    D4,D1
0032'  E741           30       ASL.W     #3,D1
0034'  9245           31       SUB.W     D5,D1
0036'  D241           32       ADD.W     D1,D1
0038'  0641 0011      33       ADDI.W    #17,D1
003C'  3C01           34       MOVE.W    D1,D6
                      35
003E'  611A           36       BSR.S     AUSKI
                      37
0040'  9A46           38 LLOE  SUB.W     D6,D5
0042'  6A0A           39 LOE2  BPL.S     NOTAD
0044'  D87C 0001      40       ADD.W     #1,D4
0048'  DA7C 0008      41       ADD.W     #8,D5
004C'  60F4           42       BRA.S     LOE2
004E'  0BB3 4000      43 NOTAD BCLR      D5,0(A3,D4.W)
0052'  0C44 01F5      44       CMPI.W    #FIN+1,D4
0056'  65E8           45       BCS.S     LLOE
0058'  60B6           46       BRA.S     SUZL
                      47
```

P.19.2 Das Sieb des Eratosthenes (I) für QL

byte, hole ein Byte) angewählt. Auf timeout wurde mit −1 verzichtet. Wurde keine Taste gedrückt, enthält nach Aufruf mit TRAP #3 D1 den Wert 0 (geprüft mit TST), und es geht zurück in die Warteschleife TASTE, sonst RTS.

```
005A'   207C 0001 0001  62 AUSKI  MOVEA.L   #SCHI,A0
0060'   3478 00CE       63        MOVEA.W   MINT,A2
0064'   4E92            64        JSR       (A2)
0066'   7012            65 NEWLI  MOVEQ     #SDNL,D0
0068'   4E43            66        TRAP      #3
006A'   0C00 FFFC       67        CMPI.B    #-4,D0
006E'   6610            68        BNE.S     KEYIN
0070'   7018            69 SCROL  MOVEQ     #SCRL,D0
0072'   72F6            70        MOVEQ     #-10,D1
0074'   7601            71        MOVEQ     #1,D3
0076'   4E43            72        TRAP      #3
0078'   7011            73 LIRAN  MOVEQ     #TAB,D0
007A'   7200            74        MOVEQ     #0,D1
007C'   7601            75        MOVEQ     #1,D3
007E'   4E43            76        TRAP      #3
0080'   207C 0000 0000  77 KEYIN  MOVEA.L   #TAST,A0
0086'   363C FFFF       78        MOVE.W    #-1,D3
008A'   7001            79 TASTE  MOVEQ     #FBYTE,D0
008C'   4E43            80        TRAP      #3
008E'   4A01            81        TST.B     D1
0090'   67F8            82        BEQ.S     TASTE
0092'   4E75            83        RTS
                        84
0094'  =01F5            85 ZALEN  DS.B      FIN+1
                        86
                        87
                        88
                        89
                        90
                        91        END
```

P.19.2 Das Sieb des Eratosthenes (II) für QL

19.4 Das Eratosthenes-Programm für den Amiga

```
020000
= 000001F4              FIN     EQU     500
020000
020000  48E7 FFFE      ERSTIN: MOVEM.L D0-D7/A0-A6,-(A7)
020004  47FA 00AA      INIMEM: LEA     ZAHLEN(PC),A3
020008  303C 01F5              MOVE.W  #FIN+1,D0
02000C  16FC 00FF      LOPINI: MOVE.B  #$FF,(A3)+
020010  51C8 FFFA              DBRA    D0,LOPINI
020014
020014  47FA 009A      SUCHZL: LEA     ZAHLEN(PC),A3
020018  7800                  MOVEQ   #0,D4
02001A  7A07           LOPAUS: MOVEQ   #7,D5
02001C  0BB3 4000      LOPINN: BCLR    D5,0(A3,D4.W)
020020  6614                  BNE.S   GEFUND
020022  51CD FFF8              DBRA    D5,LOPINN
020026  0644 0001              ADD.W   #1,D4
02002A  0C44 01F4              CMPI.W  #FIN,D4
02002E  6BEA                  BMI.S   LOPAUS
020030  4CDF 7FFF              MOVEM.L (A7)+,D0-D7/A0-A6
020034  4E75                  RTS
020036
020036  3004           GEFUND: MOVE.W  D4,D0
020038  E740                  ASL.W   #3,D0
02003A  9045                  SUB.W   D5,D0
02003C  D040                  ADD.W   D0,D0
02003E  0640 0011             ADDI.W  #17,D0
020042  3C00                  MOVE.W  D0,D6
020044
020044  4BFA 002E              LEA     REGIST(PC),A5
020048  48D5 7FFF              MOVEM.L D0-D7/A0-A6,(A5)
02004C  4CDF 7FFF              MOVEM.L (A7)+,D0-D7/A0-A6
020050  4E75                  RTS
```

P.19.3 Das Sieb des Eratosthenes (I) für Amiga

```
020052  48E7 FFFE  ZWEIIN: MOVEM.L D0-D7/A0-A6,-(A7)
020056  4BFA 001C          LEA     REGIST(PC),A5
02005A  4CD5 7FFF          MOVEM.L (A5),D0-D7/A0-A6
02005E
02005E  9A46        LOPLOE: SUB.W   D6,D5
020060  6A06        LOPLO2: BPL.S   NOTADD
020062  5244                ADDQ.W  #1,D4
020064  5045                ADDQ.W  #8,D5
020066  60F8                BRA.S   LOPLO2
020068  0BB3 4000  NOTADD: BCLR    D5,0(A3,D4.W)
02006C  0C44 01F5          CMPI.W  #FIN+1,D4
020070  65EC                BCS.S   LOPLOE
020072  60A0                BRA.S   SUCHZL
020074
020074              REGIST: DS.L    15
0200B0              ZAHLEN: DS.B    FIN+1
0202A5
0202A5                      END
```

P.19.3 Das Sieb des Eratosthenes (II) für Amiga

Zu P.19.3 ist nur noch wenig zu sagen. Bei diesem Amiga-Programm wurde die Ausgabe nach BASIC verlegt. Dort, wo in P.19.2 BSR.S AUSKI stand, wird hier das Maschinenprogramm vorübergehend zur Realisierung der Ausgabe verlassen, und nach Rückkehr aus BASIC geht es hier weiter. Dabei sind zwei wichtige Randprobleme zu lösen:

Beim Übergang nach BASIC müssen zumindest die Inhalte der benutzten Register für den Fortgang zwischengespeichert werden; hierfür ist im Arbeitsbereich Platz reserviert: REGIST. Es scheint, daß man beim Amiga gut daran tut, bei Rückkehr nach BASIC die von dort übernommenen Registerinhalte wiederherzustellen, worauf bereits früher hingewiesen wurde.

Umgekehrt müssen beim Wiedereintritt nach ZWEIIN die «BASIC-Register» wieder zwischengespeichert und die «Programmregister» wiederhergestellt werden. Wie das gelöst wurde, dürfte leicht zu erkennen sein. Da bei Niederschrift dieser Zeilen noch kein Assembler zur Verfügung stand, wurde das Programm auf dem NDR-Computer erstellt: Wie schön, daß die M68000-Programme i.a. relokatibel sind!

```
code$=STRING$(1000,0): GOSUB holprog

CALL anfad
     zeigen: PRINT PEEKW(anfad+118)
     warten: IF MOUSE(0)=0 THEN warten
     start2= anfad+82
CALL start2: GOTO zeigen

holprog: anfad=SADD(code$)
   p$="48E7 FFFE 47FA 00AA 303C 01F5 16FC 00FF 51C8 "
p$=p$+"FFFA 47FA 009A 7800 7A07 0BB3 4000 6614 51CD "
p$=p$+"FFF8 0644 0001 0C44 01F4 6BEA 4CDF 7FFF 4E75 "
p$=p$+"3004 E740 9045 D040 0640 0011 3C00 4BFA 002E "
p$=p$+"48D5 7FFF 4CDF 7FFF 4E75 48E7 FFFE 4BFA 001C "
p$=p$+"4CD5 7FFF 9A46 6A06 5244 5045 60F8 0BB3 4000 "
p$=p$+"0C44 01F5 65EC 60A0 "
   FOR i=0 TO 57
     POKEW anfad+2*i,VAL("&H"+MID$(p$,5*i+1,4))
   NEXT i: RETURN
```

P.19.3H Das Eratosthenes-Hostprogramm für Amiga

Erfreulicherweise profitiert das Amiga-BASIC von der Potenz des Prozessors: Strings dürfen z.B. bis zu 32767 Zeichen enthalten. Das regte an, den Objektcode als String bereitzustellen; er wird von dort ähnlich wie in P.7.1H eingelesen.

Der Aufruf des ersten Programmteils und die Anzeige bedürfen keiner Erklärung. Interessant ist die Mausfunktion MOUSE(0); ist die linke Maustaste nicht gedrückt, ist MOUSE(0) = 0, und es wird gewartet, bis die Taste betätigt wird. Dann geht es sehr schnell weiter. Der Wiedereintritt in das Maschinenprogramm muß ausgezählt werden: start2 = anfad + $52.

20
Sortieren

Bubblesort (bubble = (Seifen-)Blase) ist ein recht schnelles Sortierverfahren. Stellen Sie sich einen unten mit CO_2, darüber mit Luft und oben mit He gefüllten Standzylinder vor. Läßt man eine luftgefüllte Seifenblase hineinplumpsen, sinkt sie durch des Helium und zum Teil auch durch die Luft hindurch. Irgendwo zwischen Luft und Kohlendioxid wird sie zur Ruhe kommen – bis sie platzt.

Bubblesort macht das auch so. Im Speicher stehen bereits einige Daten der Größe nach geordnet. Das neue Datum wird nun so lange durch die bereits eingespeicherten geschoben, bis es an eines gelangt, das größer ist. Dann rücken die größeren einen Platz auf, und das neue wird eingeschoben. Das letzte (Pseudo-)Datum muß dazu unerreichbar groß gemacht werden. Wir wählen dafür -1 = $FF, denn es gibt kein Schriftzeichen mit einer so hohen Codierung.

20.1 Schriftzeichen sortieren

A0 ist Pointer zum Speicherbereich, A1 ist stets auf das erste Byte hinter (das könnte man auch anders machen) dem hier benutzten Schlußzeichen -1 = $FF gerichtet. Die Eingabe erfolgt wie in P.19.1, nur daß nach @CSTS das über die Tastatur eingegebene Zeichen mit @CI (character in (?)) auch tatsächlich nach D0 geholt wird. Echo mit @CO2 (character out).

Da @CO2 offenbar den Inhalt von D0 verändert, wird die Eingabe nach D2 zwischengespeichert.

Für die auf Bildschirm oder Drucker darstellbaren Zeichen (einschl. Space) gilt i.a.:

$$\$1F < \text{ASC(Zeichen)} < \$7F.$$

Ist nach CMPI.B #$20,D0 Carryflag gesetzt, war der eingelesene Code kleiner als $20, i.a. ein CTRL-Code, z.B. CTRL-C. In diesem Falle geht es

```
020000  41FA 0044        START:  LEA      SPEICH(PC),A0
020004  10BC 00FF                MOVE.B   #-1,(A0)
020008  43E8 0001                LEA      1(A0),A1
02000C  4EB8 09D8        EINGEB: JSR      @CSTS.S
020010  67FA                     BEQ.S    EINGEB
020012  4EB8 0A00                JSR      @CI.S
020016  1400                     MOVE.B   D0,D2
020018  4EB8 0AE0                JSR      @CO2.S
02001C  1002                     MOVE.B   D2,D0
02001E  0C00 0020                CMPI.B   #$20,D0
020022  6402                     BCC.S    SUCHEN
020024  4E75                     RTS
020026
020026  41FA 001E        SUCHEN: LEA      SPEICH(PC),A0
02002A  43E9 0001                LEA      1(A1),A1
02002E  B018             SUCH2:  CMP.B    (A0)+,D0
020030  62FC                     BHI.S    SUCH2
020032  45D1             AUFRCK: LEA      (A1),A2
020034  1562 0001        AUFR2:  MOVE.B   -(A2),1(A2)
020038  B5C8                     CMPA.L   A0,A2
02003A  64F8                     BCC.S    AUFR2
02003C  1140 FFFF        HINEIN: MOVE.B   D0,-1(A0)
020040  60CA                     BRA.S    EINGEB
020042
020042  11111111                 DC.L     $11111111
020046                   SPEICH: DS.B     10000
022756
022756                           END
```

P.20.1 Schriftzeichen sortieren (NDR-Computer)

zurück ins Betriebssystem; anderenfalls muß ein Platz fürs Zeichen gesucht werden:

Der Laufpointer A0 wird auf Speicheranfang zurückgesetzt, der Ende-Pointer der gültigen Eingabe entsprechend um 1 erhöht. Nun werden die eingespeicherten .B-Daten der Reihe nach (Postinkrement) mit dem neuen Datum verglichen. Solange dieses HIgher ist, geht es zurück zum nächsten Vergleich.

Wurde endlich ein Zeichen > Eingabe festgestellt, übernimmt der Aufschiebezeiger A2 den Inhalt von A1 und zeigt damit zu dem Byte, in das das

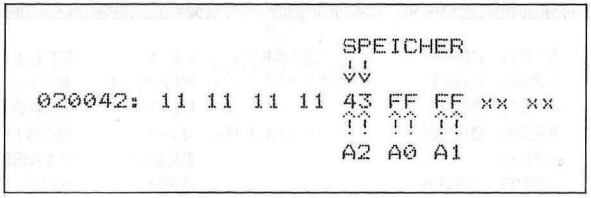


Bild 20.1 Der Speicher nach Eingabe von «C»

Schlußzeichen FF aufrücken soll. Da von unten her aufgerückt werden muß, geschieht das mit Predekrement; während des Aufschiebens zeigt A2 auf das geholte Byte, das nach (A2)+1 abzulegen ist.

Wegen des Postinkrements zeigt A0 auf das Byte oberhalb des freigemachten; Einfügen also nach −1(A0). Vorher mußte aber noch aufgerückt werden, bis A2 den Inhalt von A0 unterschritt, denn von dort war das als letztes nachzuschiebende Element zu holen. Abbruchbedingung: A2 < A0 bzw. A2 − A0 < 0 führt zum Setzen des Carryflags; zurück zu neuem Aufschieben also nur bei Carry Clear.

Bild 20.1 zeigt den Speicherzustand nach Eingabe von «C» im Augenblick des Erreichens des Sprungbefehls in 020040. Das an sich unsinnige Langwort $11111111 erleichtert die Orientierung im Speicher.

20.2 Die NDR-Routine @READ

Das Sortieren von Schriftzeichen soll in zwei Stufen zum alphabetischen Sortieren von Wörtern unterschiedlicher Länge gesteigert werden. Die Zwischenstufe ist das Sortieren von Wörtern vorgegebener Länge. In beiden Fällen spielt die Stringeinlese-Routine eine dominierende Rolle. Ihr Ablauf muß sorgfältig beobachtet werden. Beim NDR-Klein-Computer ist das die Routine @READ.

Unser Testprogramm zeigt schon den grundsätzlichen Ansatz, der im folgenden beibehalten werden soll: Der 34 Bytes lange Puffer dient als Eingabebereich. Die eingegebenen Wörter dürfen bis zu 32 Zeichen lang sein. Obgleich bis auf weiteres unnötig, soll der String mit der Angabe seiner Länge .B beginnen. Als String-Schlußzeichen verwendet das Grundprogramm das Byte 00. Zählt man dieses ebenfalls mit, so ergibt sich die Pufferlänge 34.

```
020000                  *
020000                  * P.20.2 DIE SUBROUTINE @READ
020000
020000  7014            INIT:   MOVEQ   #20,D0
020002  41FA 008E               LEA     SPEICH(PC),A0
020006  4298            LOPCLR: CLR.L   (A0)+
020008  51C8 FFFC               DBRA    D0,LOPCLR
02000C  45FA 0084               LEA     SPEICH(PC),A2
020010  34BC 01FF               MOVE.W  #$01FF,(A2)
020014  426A 0002               CLR.W   02(A2)
020018  47EA 0004               LEA     04(A2),A3
02001C
02001C  41FA 004F       EINGEB: LEA     BUFFER+1(PC),A0
020020  43FA 001A               LEA     READPA(PC),A1
020024  4C91 000F               MOVEM.W (A1),D0-D3
020028  4EB8 1486               JSR     @READ.S
02002C  41FA 003E               LEA     BUFFER(PC),A0
020030  1084                    MOVE.B  D4,(A0)
020032
020032  49FA 0010       RETTEN: LEA     REGIST(PC),A4
020036  48D4 1F1F               MOVEM.L D0-D4/A0-A4,(A4)
02003A  4E75                    RTS
02003C
02003C  00110002        READPA: DC.L    $00110002
020040  00800020                DC.L    $00800020
020044                  REGIST: DS.L    10
02006C                  BUFFER: DS.B    34
02008E  2A2A2A2A                DC.B    '****'
020092                  SPEICH: DS.B    100*34
020DDA
020DDA                          END
```

P.20.2 Die Subroutine @READ

An den Eingabebereich anschließend stehen vier Sternchen im Speicher – lediglich eine Test- und Orientierungshilfe. An ihn schließt sich der (Daten-) Speicher an, der für die Aufnahme von 100 Wörtern dimensioniert wird. Beim NDR-Computer ist diese Vorgabe am Ende des Programms unkritisch; anders bei Computern wie QL oder Amiga, bei denen die Daten in das nächste Programm, in Systemvariable oder in den Jobkontrollblock (s. Glossar) hineinwachsen könnten.

Das Programm beginnt mit Nullsetzen des Speichers, was die Übersicht fördert. Vorn im Speicher steht der Eintrag 01FF0000, als Zeichen für Vergleichsschluß wieder der unerreichbare Code FF; da er ein Byte lang ist, geht ihm 01 voran, gefolgt vom Stringende-Symbol 00, das hier sogar doppelt gesetzt wurde.

Vor der Eingabe wird A0 auf den Puffer gerichtet, genauer: auf Buffer+1, um Platz für die Längenangabe zu lassen. Dann werden die bereitgestellten Eingabeparameter (READPA) geladen: D0 = $11 bestimmt die kleinste Schriftgröße, Breite 1 und Höhe 1. Das Eingabefensterchen wird bei x = 2 und y = 128 (Pixels) aufgemacht und bietet Platz für bis zu D3 = 32 Zeichen. Kürzere Eingaben müssen mit cr abgeschlossen werden. Gibt man im Testlauf ein 32buchstabiges Wort ein, wird die Eingabe abrupt, aber ordnungsgemäß abgebrochen, und im Puffer stehen $20, alle eingetippten Zeichen und 00. D4 enthält anschließend die Anzahl der gültigen eingegebenen Zeichen. @READ hängt das Schlußzeichen 00 automatisch an den String an.

Vor dem Rücksprung ins Betriebssystem werden die beteiligten Register im Speicher abgelegt, was eine bequeme Kontrolle bei der weiteren Programmentwicklung ermöglichen soll.

```
Buffer:    09 57 55 45 52 5A 42 55 52 47 00 xx xx
              W  U  E  R  Z  B  U  R  G

Speicher: 01 FF 00 00 oo oo oo oo oo oo oo oo oo
                         ^
                        A3

          D0 = 00000011      A0 = 0002006C
          D1 = 00000002      A1 = 0002003C
          D2 = 00000080      A2 = 00020092
          D3 = 00000020      A3 = 00020096
          D4 = 00000009      A4 = 00020044
```

Bild 20.2 Speicherauszug nach Eingabe von WUERZBURG

20.3 Sortieren von Wörtern gleicher Maximallänge

In diesem Abschnitt dürfen die Wörter bis zu 32 gültige Zeichen lang sein; mit vorangestellter Längenangabe und dem Schlußbyte werden je Wort im Speicher 34 Bytes belegt. Wir nennen ein derartiges 34-Byte-Feld (später auch kürzere) Record (Aufzeichnung).

```
020000                  * ========================================== *
020000                  * BLOCK 1
020000
020000  7050            INIT:   MOVEQ   #80,D0
020002  41FA 00D4               LEA     SPEICH(PC),A0
020006  4298            LOPCLR: CLR.L   (A0)+
020008  51C8 FFFC               DBRA    D0,LOPCLR
02000C  45FA 00CA               LEA     SPEICH(PC),A2
020010  34BC 01FF               MOVE.W  #$01FF,(A2)
020014  426A 0002               CLR.W   02(A2)
020018  45EA 0022               LEA     34(A2),A2
02001C  47D2                    LEA     (A2),A3
02001E
02001E  41FA 0093 EINGEB: LEA   BUFFER+1(PC),A0
020022  43FA 0086               LEA     READPA(PC),A1
020026  4C91 000F               MOVEM.W (A1),D0-D3
02002A  4EB8 1486               JSR     @READ.S
02002E  41FA 0082               LEA     BUFFER(PC),A0
020032  1084                   MOVE.B  D4,(A0)
020034  6700 004E               BEQ     AUSGEB
020038
020038                  * A3 -> NXTFRE

020038                  * ========================================== *
020038                  * BLOCK 2
020038
020038  43FA 009E VERGL:  LEA     SPEICH(PC),A1
02003C  45D1      VERGL2: LEA     (A1),A2
02003E  41FA 0072               LEA     BUFFER(PC),A0
020042  1018                   MOVE.B  (A0)+,D0
020044  121A                   MOVE.B  (A2)+,D1
020046  B200                   CMP.B   D0,D1
020048  6502                   BCS.S   VERGL3
02004A  C141                   EXG     D0,D1
02004C  4880      VERGL3: EXT.W   D0
02004E  5340                   SUBQ.W  #1,D0
```

```
020050  B508          VERGL4: CMPM.B   (A0)+,(A2)+
020052  620C                  BHI.S    AUFSCH
020054  6604                  BNE.S    VERGL5
020056  51C8 FFF8             DBRA     D0,VERGL4
02005A  43E9 0022     VERGL5: LEA      34(A1),A1
02005E  60DC                  BRA.S    VERGL2
020060
020060                * A1 -> GEPREC

020060                * ================================ *
```

P.20.3 Sortieren von Wörtern gleicher Maximallänge (I)

P.20.3 beginnt ähnlich wie P.20.2, nur ist hier der Record mit dem Schlußzeichen ebenfalls 34 Bytes lang. A3 zeigt auf das erste Byte des noch nicht vorhandenen nächsten (freien) Eintrags: NXTFRE. (Da A3 diese Funktion das ganze Programm hindurch ausübt, könnte man es als «globalen Pointer» bezeichnen.) Auch die Eingabe gleicht P.20.2, nur wird beim Vorlegen des Längenbytes geprüft, ob es 00 ist, was ein Zeichen dafür wäre, daß ohne eigentliche Eingabe cr gedrückt wurde. In diesem Falle ginge es weiter zur alphabetischen Ausgabe der sortierten Wörter (Bild 20.3).

```
a)   BANANE        alt < neu
     MILCH         neu
     ^

b)   MIETE         alt < neu
     MILCH         neu
       ^

c)   MINISTER      alt > neu
     MILCH         neu
       ^
```

Bild 20.3 Verschiedene Vergleichsergebnisse

Block 2 vergleicht das im Puffer stehende Wort der Reihe nach mit dem ersten, zweiten usw. Record im Speicher. Während mit dem ersten Record verglichen wird, zeigt A1 auf dessen Anfang (GEPREC = GEPrüfter RECord) und wird beim Fortgang der Prüfung um jeweils 34 erhöht. A1 nimmt seinen Inhalt nach Block 3 mit.

Innerer Pointer ist A2; er zeigt innerhalb eines Records auf das jeweils geprüfte Zeichen. Parallel dazu läuft im Puffer A0. Zuerst werden die beiden Längen nach D0 und D1 geholt; die *größere* wird um eins vermindert zum Schleifenzähler beim Zeichenvergleich. (EXT.W erweitert (extend) das Längenbyte zum .W-Zähler.) Macht man – scheinbar paradox – die größere Länge zum Schleifenzähler, so vermeidet man damit, daß beispielsweise BIRNENMARMELADE vor BIRNE einsortiert wird, denn an der sonstigen Abbruchstelle ist $4E > 00.

Bei VERGL4 beginnt der Vergleich der gültigen Zeichen. Ist $\text{Zeichen}_{\text{Rec}} > \text{Zeichen}_{\text{Buf}}$, so ist der geprüfte Record «alphabetisch größer» als das neue Wort. BHI führt zum Auf- und anschließenden Einschieben. Im Falle $\text{Zeichen}_{\text{Rec}} \neq \text{Zeichen}_{\text{Buf}}$ liegt in Wahrheit «<» vor. Hier wird die Prüfung abgebrochen, und der Recordpointer A1 wird um 34 erhöht.

Block 2 hat nur einen einzigen Ausgang: Wenn der Platz für das neue Wort gefunden wurde, geht es weiter nach AUFSCH.

A.20.1: Wo wird ein neues Wort eingetragen, das einem vorhandenen Record gleicht?

```
020060                  * ================================== *
020060                  * BLOCK 3
020060
020060  200B            AUFSCH:  MOVE.L   A3,D0
020062  9089                     SUB.L    A1,D0
020064  5340                     SUBQ.W   #1,D0
020066  45D3                     LEA      (A3),A2
020068  1562 0022       LAUFSH:  MOVE.B   -(A2),34(A2)
02006C  51C8 FFFA               DBRA     D0,LAUFSH
020070
020070  41FA 0040       EINSCH:  LEA      BUFFER(PC),A0
020074  7021                     MOVEQ    #33,D0
020076  12D8            LEINSH:  MOVE.B   (A0)+,(A1)+
020078  51C8 FFFC               DBRA     D0,LEINSH
02007C  47EB 0022               LEA      34(A3),A3
020080  6000 FF9C               BRA      EINGEB
020084
020084                  * ================================== *
020084                  * BLOCK 4
020084
020084  4EB8 3FEC       AUSGEB:  JSR      @CLRSCREEN.S
020088  43FA 002D                LEA      SPEICH-33(PC),A1
```

```
020088C  43E9 0022 NXTWRT: LEA      34(A1),A1
020090  41D1            LEA      (A1),A0
020092  0C10 00FF        CMPI.B   #$FF,(A0)
020096  6710            BEQ.S    GRUPRO
020098  1018       NXTCHR: MOVE.B   (A0)+,D0
02009A  6706             BEQ.S    NEWLIN
02009C  4EB8 0AE0        JSR      @CO2.S
0200A0  60F6             BRA.S    NXTCHR
0200A2  4EB8 1698  NEWLIN: JSR      @CRLF.S
0200A6  60E4             BRA.S    NXTWRT
0200A8  4E75       GRUPRO: RTS
0200AA
0200AA             * ================================ *
0200AA             * BLOCK 5
0200AA
0200AA  00110002   READPA: DC.L     $00110002
0200AE  00800020           DC.L     $00800020
0200B2             BUFFER: DS.B     34
0200D4  2A2A2A2A           DC.B     '****'
0200D8             SPEICH: DS.B     3404
020E24

                           END
```

P.20.3 Sortieren von Wörtern gleicher Maximallänge (II)

Zu Block 3: Da jeder Record 34 Bytes lang ist, muß zum Platz schaffen um 34 Bytes aufgerückt werden, was MOVE.B −(A2),34(A2) bewirkt. Nehmen wir an, das neue Wort sei ganz hinten als letztes einzuordnen, es käme an die Stelle, an der sich momentan das Schlußzeichen befindet. Dann zeigt A1 auf das Schlußzeichen und A3 auf die Stelle NXTFRE, zu der es aufgeschoben werden soll. Sozusagen per Definition wäre hier A3 − A1 = 34, also die Anzahl der aufrückenden Bytes. Wären der letzte Record und das Schlußzeichen zu verschieben, wäre A3 − A1 = 68 usw. Da in einer DBRA-Schleife verschoben wird, ist der Schleifenzähler D0 auf A3 − A1 − 1 zu setzen.

A.20.2: Betrachten Sie noch einmal den Befehl MOVE.B −(A2),34(A2).
a) Wird durch das Predekrement die Verschiebung um 34 gestört?
b) Könnte man auch mit Postinkrement verschieben?

Das Einschieben des neuen Worts in den freikopierten Record ist reine Routinearbeit; abschließend wird A3 um 34 erhöht. Zur Ausgabe ist nur noch zu vermerken, daß bei Feststellung des Schlußzeichens nicht mehr ausgegeben, sondern ins Grundprogramm zurückgesprungen wird.

20.4 Sortieren von Wörtern individueller Länge

P.20.3 bot neben dem Vorteil einer etwas besseren Durchschaubarkeit zwei Nachteile:

a) Es wird Speicherplatz verschwendet.
b) Die Länge der Records ist begrenzt. P.20.4 soll das verbessern. Bild 20.4 zeigt einen Speicherauszug bei gepackter Unterbringung von Wörtern individueller Länge. (Dabei wurde das Schlußzeichen FF lesbarkeitshalber durch «*» wiedergegeben.)

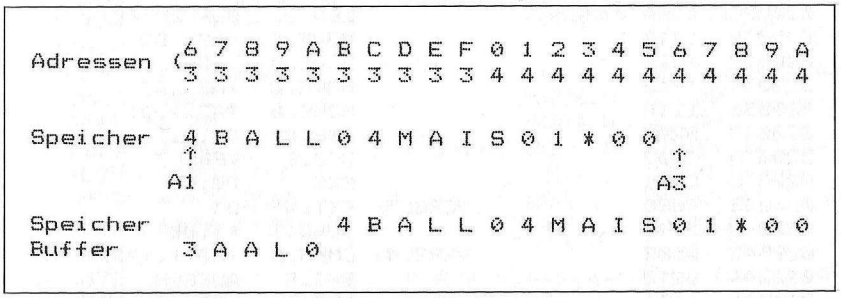


Bild 20.4 Der Speicher bei gepackter Belegung

(Da zum Lieferumfang der benutzten Maschinen bereits Textverarbeitungsprogramme gehören, wurde auf das ursprüngliche Vorhaben, ein solches darzustellen, verzichtet. Der Grundansatz entspricht aber dem von P.20.4: Wenn die Unterbringung des Textes im Speicher geklärt ist, ergibt sich alles andere fast von allein.)

Sechserlei wird nun anders:

1. Pointer A3 zeigt unmittelbar hinter das Schlußzeichen (NXTFRE), rückt also im Vergleich zum Bisherigen um 30 Positionen auf.
2. Die Nettolänge des neuen Eintrags wird von D2 festgehalten.

```
020000                          * ================================ *
020000                          * BLOCK 1
020000
020000  45FA 0112        SORTIR: LEA      SPEICH(PC),A2
020004  34FC 01FF                MOVE.W   #$01FF,(A2)+
020008  425A                     CLR.W    (A2)+
02000A  47D2                     LEA      (A2),A3
02000C
02000C  41FA 00A3        EINGEB: LEA      BUFFER+1(PC),A0
020010  43FA 0096                LEA      READPA(PC),A1
020014  4C91 000F                MOVEM.W  (A1),D0-D3
020018  4EB8 1486                JSR      @READ.S
02001C  41FA 0092                LEA      BUFFER(PC),A0
020020  1084                     MOVE.B   D4,(A0)
020022  6700 005A                BEQ      AUSGEB
020026
020026                          * A3 -> NXTFRE

020026                          * ================================ *
020026                          * BLOCK 2
020026
020026  43FA 00EC        VERGL:  LEA      SPEICH(PC),A1
02002A  45D1             VERGL2: LEA      (A1),A2
02002C  41FA 0082                LEA      BUFFER(PC),A0
020030  1410                     MOVE.B   (A0),D2
020032  1018                     MOVE.B   (A0)+,D0
020034  1612                     MOVE.B   (A2),D3
020036  121A                     MOVE.B   (A2)+,D1
020038  B200                     CMP.B    D0,D1
02003A  6502                     BCS.S    VERGL3
02003C  C141                     EXG      D0,D1
02003E  4880             VERGL3: EXT.W    D0
020040  5340                     SUBQ.W   #1,D0
020042  B508             VERGL4: CMPM.B   (A0)+,(A2)+
020044  6210                     BHI.S    AUFSCH
020046  6604                     BNE.S    VERGL5
020048  51C8 FFF8                DBRA     D0,VERGL4
02004C  4882             VERGL5: EXT.W    D2
02004E  4883                     EXT.W    D3
020050  43F1 3002                LEA      02(A1,D3.W),A1
020054  60D4                     BRA.S    VERGL2
020056
020056                          * D2 =  LENNEU
020056                          * D3 =  LENALT
020056                          * A1 -> GEPREC

020056                          * ================================ *
```

P.20.4 Sortieren von Wörtern individueller Länge (I)

3. Die Nettolänge des momentan verglichenen Records erfaßt D3.
4. Zum Vergleich des nächsten Records ist A1 um D3 + 2 zu erhöhen.
5. Aufgerückt wird (A3 + D2 − A1)-mal um (D2 + 2) Bytes.
6. Nach dem Aufrücken sind nur noch D2 + 2 Bytes einzuschieben.

Außerdem wurden unnötige Diagnosehilfen weggelassen.

```
020056                          *
020056                          * ================================ +
020056                          * BLOCK 3
020056
020056  200B                    AUFSCH: MOVE.L   A3,D0
020058  D082                            ADD.L    D2,D0
02005A  9089                            SUB.L    A1,D0
02005C  5340                            SUBQ.W   #1,D0
02005E  45D3                            LEA      (A3),A2
020060  15A2 2002              LAUFSH: MOVE.B   -(A2),02(A2,D2.W)
020064  51C8 FFFA                       DBRA     D0,LAUFSH
020068
020068  41FA 0046              EINSCH: LEA      BUFFER(PC),A0
02006C  3002                            MOVE.W   D2,D0
02006E  5240                            ADDQ.W   #1,D0
020070  12D8                  LEINSH: MOVE.B   (A0)+,(A1)+
020072  51C8 FFFC                       DBRA     D0,LEINSH
020076  47F3 2002                       LEA      02(A3,D2.W),A3
02007A  6000 FF90                       BRA      EINGEB
02007E
02007E                          * ================================ +
02007E                          * BLOCK 4
02007E
02007E  4EB8 3FEC              AUSGEB: JSR      @CLRSCREEN.S
020082  43FA 0090                       LEA      SPEICH(PC),A1
020086  1419                  NXTWRT: MOVE.B   (A1)+,D2
020088  4882                          EXT.W    D2
02008A  41D1                          LEA      (A1),A0
02008C  0C10 00FF                     CMPI.B   #$FF,(A0)
020090  6714                          BEQ.S    GRUPRO
020092  1018                  NXTCHR: MOVE.B   (A0)+,D0
020094  6706                          BEQ.S    NEWLIN
020096  4EB8 0AE0                     JSR      @CO2.S
02009A  60F6                          BRA.S    NXTCHR
02009C  4EB8 1698              NEWLIN: JSR      @CRLF.S
0200A0  43F1 2001                     LEA      01(A1,D2.W),A1
```

```
0200A4  60E0                    BRA.S   NXTWRT
0200A6
0200A6  4E75            GRUPRO: RTS
0200A8
0200A8                  * D2 =  NETLEN
0200A8                  * ==========================================*
0200A8                  * BLOCK 5
0200A8
0200A8  00110002        READPA: DC.L    $00110002
0200AC  00800020                DC.L    $00800020
0200B0                  BUFFER: DS.B    100
020114                  SPEICH: DS.B    3404
020E60
020E60                          END
```

P.20.4 Sortieren von Wörtern individueller Länge (II)

21
Großes Sortierprogramm

Ernsthafte Sortierjobs erstrecken sich häufig über mehrere Tage, das Fortschreiben einer Adreßdatei über Jahre. Es muß also möglich sein, Dateien auf Diskette abzulegen, von dort zurückzuholen, einzelne Records zu löschen, wahlweise auf Drucker oder Bildschirm auszugeben. Immer wieder manifestiert sich auch hier, daß ein Computer nicht besser sein kann als die mitgelieferte oder zu angemessenen Preisen erhältliche Dokumentation.

Obgleich der QL recht ordentlich dokumentiert ist, wurde wegen des transparenteren Betriebssystems und des besseren Entwicklungskomforts hier wieder einmal der NDR-Computer mit Systema-DOS vorgezogen. Die folgenden Programmteile P.21.1 bis P.21.7 gehören zusammen und bilden gemeinsam P.21.

21.1 Das Menü

Beim Programmstart wird ab MENUE2 der Bildschirm gelöscht und das Menü ausgegeben. Die bereits mehrmals benutzte Ausgaberoutine wurde hier (s.u.) ein wenig an die speziellen Erfordernisse dieses Programms angepaßt. (Der unterbrochene Schrägstrich in den Texten steht für «ö».) Die Texte wurden wieder nach unserer P.2x-Norm codiert. Am Ende steht das Schlußzeichen 01FF0000; weil es während der Programmentwicklung gelegentlich ungeraden Adressen zugewiesen wurde, wird es hier in einzelne Bytes zerlegt. Die Assemblerdirektive DS.B 0 reserviert formal gedeutet 0 Bytes; steht sie aber auf ungeradem Platz, reserviert sie 1 Byte und sorgt somit dafür, daß der nächste Befehl wieder auf einer geraden Adresse eingetragen wird. (Man findet hierfür auch Direktiven wie EV oder EVEN geradzahlig.) Das Menü kann aber auch aufgerufen werden über Label MENUE. In diesem Falle verharrt die Routine KEYIN in einer Warteschleife, bis die Taste «M» gedrückt wird. Das ist sinnvoll, wenn eine Ausgabe nicht sofort wieder gelöscht werden soll (s.u.).

```
020000                          * P.21   GROSSES SORTIERPROGRAMM
020000                          *
020000
020000  600A                    START:  BRA.S    MENUE2
020002  6100 00DE                MENUE:  BSR      KEYIN
020006  0C00 004D                        CMPI.B   #'M',D0
02000A  66F6                            BNE.S    MENUE
02000C  4EB9 00003FEC           MENUE2: JSR      @CLRSCREEN
020012  43FA 000A                MENUE3: LEA      TEXTE(PC),A1
020016  6100 024C                        BSR      AUSGB3
02001A  6000 008A                        BRA      OPTION
02001E  40 3029204CFC           TEXTE:  DC.B  64,'0) L/sch '
020024  73636820
020028  616C6C65732020                  DC.B       'alles  '
02002F  33292041627370                  DC.B     '3) Abspei'
020036  6569
020038  636865726E2020                  DC.B       'chern  '
02003F  35292041E7A65                   DC.B     '5) Anzeig'
020046  6967
020048  656E2020202020                  DC.B       'en     '
02004F  32292045696E67                  DC.B     '2) Eingeb'
020056  6562
020058  656E2020202020                  DC.B       'en     ',0
02005F  00
020060  40 3129204CFC                   DC.B  64,'1) L/sch '
020066  73636820
02006A  65696E73202020                  DC.B       'eins   '
020071  3429204C616465                  DC.B     '4) Laden '
020078  6E20
02007A  20202020202020                  DC.B       '       '
020081  36292044727563                  DC.B     '6) Drucke'
020088  6B65
02008A  6E2020202020                    DC.B       'n      '
020091  372920456E6465                  DC.B     '7) Ende  '
020098  2020
02009A  20202020202020                  DC.B       '       ',0
0200A1  00
0200A2  01 FF 00 00                     DC.B     1,$FF,0,0
0200A6                                  DS.B     0
0200A6
```

P.21.1 Das Menü

21.2 Die Auswahl eines Menüpunkts

```
0200A6  613A              OPTION: BSR.S   KEYIN
0200A8  0200 0007                 ANDI.B  #7,D0
0200AC  E500                      ASL.B   #2,D0
0200AE  4880                      EXT.W   D0
0200B0  41FA 0010                 LEA     OPTADR(PC),A0
0200B4  2070 0000                 MOVEA.L 0(A0,D0.W),A0
0200B8  4E90                      JSR     (A0)
0200BA  6000 FF50                 BRA     MENUE2
0200BE  6000 FF42                 BRA     MENUE
0200C2
0200C2  000200F2          OPTADR: DC.L    ALLWEG
0200C6  00020110                  DC.L    TILGEN
0200CA  00020192                  DC.L    SORTIR
0200CE  00020224                  DC.L    SAVEN
0200D2  00020248                  DC.L    LADEN
0200D6  00020258                  DC.L    AUSGEB
0200DA  0002028A                  DC.L    DRUKEN
0200DE  0002029C                  DC.L    GRUPRO
0200E2
0200E2  4EB9 000009D8    KEYIN:   JSR     @CSTS
0200E8  67F8                      BEQ.S   KEYIN
0200EA  4EB9 00000A00             JSR     @CI
0200F0  4E75                      RTS
```

P.21.2 Auswahl aus dem Menü

Die acht Menüpunkte (Optionen) sind von 0 bis 7 numeriert; die Auswahl erfolgt über KEYIN, was dem GET mancher BASIC-Dialekte entspricht. Der ASCII des eingetippten Zeichens steht in D0 und wird dort ohne weitere Kontrolle auf drei Bits zurückmaskiert. Die verbliebene einziffrige Zahl wird vervierfacht und auf .W-Format erweitert. So kann sie als Index bei der Auswahl der richtigen Einsprungadresse dienen.

Der Befehl MOVEA verdient eine kurze Würdigung. Vor dem Komma steht die Quelladresse; war D0 = 3, so ist diese Quelladresse 200CE. Da hier nicht die Konstantenadressierung benutzt wird, wird A0 nicht mit 000200CE geladen, sondern mit der .L-Adresse, die in 200CE eingeschrieben ist, also wird A0 mit 20224 geladen, der Einsprungadresse der Option

SAVEN. Das aus den acht möglichen Unterprogrammen ausgewählte wird nun mit JSR (A0) als Subroutine aufgerufen. Die Unterprogramme müssen also mit RTS enden. Im Regelfall wird anschließend der Programmablauf bei MENUE2 mit Löschen des Bildschirms und Menüausgabe fortgesetzt. Wird die Rücksprungadresse aber von der aufgerufenen Routine im Stapel um 4 erhöht, erfolgt der Rücksprung nach MENUE mit Warten auf die Löschfreigabe durch Taste «M».

A.21.1: Wo werden Adressen im Stapel manipuliert?

Zur Erläuterung von KEYIN sei daran erinnert, daß @CSTS fragt, ob eine Taste gedrückt wurde. Wenn nicht, ist D0 leer; eine Abfrage von D0 erübrigt sich jedoch, weil das RTS der Systemroutine das Z-Flag nicht beeinflußt. Wurde jedoch eine Taste gedrückt, holt @CI ihren Code nach D0. RTS.

21.3 Alles oder einzelne Records löschen

ALLWEG löscht übersichtlichkeitshalber den Speicheranfang, setzt die Pointer zurück und setzt unser Schlußzeichen 01FF0000 an den Speicheranfang. Neu an P.21 ist u. a., daß der jeweilige Inhalt des End-Pointers A3 unmittelbar vor dem ersten Record in PNTEND abgelegt wird. So wird es möglich, ihn mit auf Diskette zu saven oder von dort zurückzuholen und dann die Records fortzuschreiben.

ALLWEG richtet A4 auf PNTEND und A2 auf SPEICH. Nach dem Setzen des Schlußzeichens zeigt A2 auf NXTFRE; diese Adresse wird dem NXTFRE-Pointer A3 übergeben und über A4 in PNTEND abgelegt. NXTFRE wird auch an A0 weitergegeben und zur Nullsetzung des Speicheranfangs benutzt.

A.21.2: Wie viele Bytes werden gelöscht?

Option 1 (TILGEN) entfernt einen Record aus dem Speicher. Vier Abschnitte sind zu erwähnen: EINGAB, eine im Prinzip schon bekannte Subroutine, ist bei Option 2 untergebracht.

COMPA vergleicht den eingegebenen String mit den im Speicher stehenden. Dieser Teil ist schon von P.20.4 her bekannt; er kommt – ebenfalls als VERGL – in seiner Grundform auch in Option 2 vor. Erreicht der Vergleich

```
0200F2                          * ================================
0200F2                          * OPTION 0
0200F2
0200F2  49FA 0212       ALLWEG: LEA     PNTEND(PC),A4
0200F6  45FA 0212               LEA     SPEICH(PC),A2
0200FA  24FC 01FF0000           MOVE.L  #$01FF0000,(A2)+
020100  7032                    MOVEQ   #50,D0
020102  41D2                    LEA     (A2),A0
020104  47D2                    LEA     (A2),A3
020106  288B                    MOVE.L  A3,(A4)
020108  4298            LPALWG: CLR.L   (A0)+
02010A  51C8 FFFC               DBRA    D0,LPALWG
02010E  4E75                    RTS
020110
020110                          * ================================
020110                          * OPTION 1
020110
020110  6100 0086       TILGEN: BSR     EINGAB
020114  43FA 01F4       COMPA:  LEA     SPEICH(PC),A1
020118  45D1             COMPA2: LEA     (A1),A2
02011A  41FA 0186               LEA     BUFFER(PC),A0
02011E  1410                    MOVE.B  (A0),D2
020120  1018                    MOVE.B  (A0)+,D0
020122  1612                    MOVE.B  (A2),D3
020124  121A                    MOVE.B  (A2)+,D1
020126  B200                    CMP.B   D0,D1
020128  6502                    BCS.S   COMPA3
02012A  C141                    EXG     D0,D1
02012C  4880            COMPA3: EXT.W   D0
02012E  5340                    SUBQ.W  #1,D0
020130  B508            COMPA4: CMPM.B  (A0)+,(A2)+
020132  622E                    BHI.S   FEHLER
020134  6606                    BNE.S   COMPA5
020136  51C8 FFF8               DBRA    D0,COMPA4
02013A  600A                    BRA.S   HINAUS
02013C  4882            COMPA5: EXT.W   D2
02013E  4883                    EXT.W   D3
020140  43F1 3002               LEA     02(A1,D3.W),A1
020144  60D2                    BRA.S   COMPA2
```

P.21.3 Alles oder einzelne Records löschen

eine Aufzeichnung, die im Alphabet nach («HIgher») dem Wort kommt, das getilgt werden soll, liegt insofern ein Fehler vor, als das gesuchte Wort gar nicht vorhanden ist (weiter im nächsten Teillisting). Steht der momentan verglichene Record vor dem zu entfernenden im Alphabet, führt BNE zum Vergleich mit dem nächsten Eintrag. So kann die Vergleichsschleife nur über DBRA verlassen werden: In diesem Falle liegt Gleichheit vor, und das eingegebene Wort kann HINAUSgeworfen werden.

21.4 Das eigentliche Entfernen eines Worts

```
020146  200B             HINAUS: MOVE.L   A3,D0
020148  908A                     SUB.L    A2,D0
02014A  4884                     EXT.W    D4
02014C  4444                     NEG.W    D4
02014E  159A 40FD        LPTILG: MOVE.B   (A2)+,-3(A2,D4.W)
020152  51C8 FFFA                DBRA     D0,LPTILG
020156  47F3 40FE                LEA      -2(A3,D4.W),A3
02015A  49FA 01AA                LEA      PNTEND(PC),A4
02015E  288B                     MOVE.L   A3,(A4)
020160  4E75                     RTS
020162
020162  41FA 0028        FEHLER: LEA      WRITPA(PC),A0
020166  4C90 0007                MOVEM.W  (A0),D0-D2
02016A  41FA 000C                LEA      TILGTX(PC),A0
02016E  4EB9 00001386            JSR      @WRITE
020174  5897                    ADDQ.L   #4,(A7)
020176  4E75                    RTS
020178
020178  697374206E6963 TILGTX: DC.B     'ist nicht '
02017F  687420
020182  766F7268616E64          DC.B     'vorhanden',0
020189  656E00
02018C  00220002        WRITPA: DC.L     $00220002
020190  00D0                     DC.W     $00D0
```

P.21.4 Das Entfernen eines Records

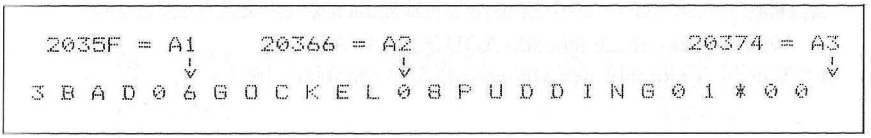


Bild 21.1 Angenommener Speicherinhalt, GOCKEL soll entfernt werden, «*» steht für FF

Angenommen, das Wort GOCKEL soll aus dem Speicher entfernt werden (Bild 21.1); dann zeigt nach dem Suchlauf A1 auf die zugehörige Längenangabe, A2 auf die Schluß-Null und A3 auf NXTFRE. Noch von der Eingabe (@READ) her enthält D4 die Anzahl der gültigen Zeichen, nämlich 6. Nun sind drei Fragen zu klären:

a) Um wie viele Bytes ist zu verschieben?
b) Wie viele Bytes sind zu verschieben?
c) Müssen die Bytes mit den höheren oder die mit den niedrigeren Adressen zuerst verschoben werden?

Zu c) Aus der Überlegung von L.20.2b – jedoch mit umgekehrter Argumentation – folgt, daß die Verschiebung bei den niedrigeren Adressen beginnen muß.

Zu a) Das Wort GOCKEL, sein Längenbyte und sein Schlußzeichen sind zu überschreiben, also muß um D4 + 2 Bytes verschoben werden; das bewirkt der Befehl MOVE.B (A2)+,−3(A2,D4.W). Das Postinkrement beim Holen sorgt für das Fortschreiten zu den höheren Adressen hin. Da dabei der Inhalt von A2 beim Ablegen schon um 1 größer geworden ist, muß um D4 + 3 zurückverschoben werden. Hierzu wurde D4 nach Erweiterung auf .W-Format negiert.

Zu b) Die Anzahl der zu verschiebenden Bytes errechnet sich aus A3 − A2 = 20374 − 20366 zu 14. Da die DBRA-Schleife beim Zählerstand 0 noch einmal durchlaufen wird, ist die Differenz um 1 zu verringern.

A.21.3: Weshalb wurde die Differenz dennoch nicht um 1 vermindert?

Nach der Verschiebung wird der NXTFRE-Pointer um −(D4 + 2) aktualisiert und in PNTEND abgelegt. Die Fehlermeldung erscheint unterhalb des Menüs.

A.21.4:
a) Was soll das abschließende ADDQ.L #4,(A7)?
b) Welche Taste gibt den weiteren Programmablauf frei?

21.5 Eingabe und Einfügen neuer Wörter

An sich ist zu diesem Programmteil wenig zu sagen, denn er wurde ja in Kapitel 20 in vier Schritten entwickelt; daher nur zur Anpassung an den neuen Rahmen:

Um die Eingaberoutine auch in Option 1 benutzen zu können, wurde sie – in bezug auf SORTIR – als Subroutine formuliert. Schon in P.20.3 und 4 wurde die Eingabe- und Einsortierschleife erst verlassen, wenn cr gedrückt, also kein gültiges Zeichen eingegeben wurde. In diesem Falle wird hier nicht bei SORTI2 weitergemacht, sondern es wird um eine Ebene weiter zurückgesprungen.

A.21.5:
a) Welcher Befehl manipuliert den Rücksprung?
b) Wohin wird zurückgesprungen?

Die Eingaberoutine hat zwei Einsprungstellen: Bei der ersten Eingabe geht es mit BSR nach EINGAB, wo neben A2 auch der NXTFRE-Pointer A3 aktualisiert wird. Das ist bei den weiteren Eingaben nicht mehr nötig; deshalb wird bei diesen nach EINGEB zurückgekehrt.

Eine gültige Eingabe bewirkt eine Programmfortsetzung bei SORTI2, von wo aus es – wie bekannt – zum Vergleich geht.

Nachdem endlich ein neues Wort eingefügt wurde, geht es aus Teil 2 (AUFSCH) der Subroutine VERGL nicht etwa mit RTS zum Hauptprogramm SORTIR zurück, sondern mit BRA sofort in die vorher durchlaufene Subroutine (EINGEB) der gleichen Ebene. Da der Einsprung beim Label SORTI2 erfolgte, wird der Programmablauf mit BRA.S SORTI2 fortgesetzt und führt wieder nach VERGL.

A.21.6: Wie oft und mit welcher Rücksprungadresse wird das RTS in 201C0 durchlaufen
a) bei Eingabe eines zu TILGENden Worts?
b) bei Eingabe von drei Wörtern und einem «einsamen» cr?

```
020192  6104            SORTIR: BSR.S   EINGAB
020194  6134            SORTI2: BSR.S   VERGL
020196  60FC                    BRA.S   SORTI2
020198
020198  49FA 016C       EINGAB: LEA     FNTEND(PC),A4
02019C  2654                    MOVEA.L (A4),A3
02019E  45FA 016A               LEA     SPEICH(PC),A2
0201A2
0201A2  41FA 00FF       EINGEB: LEA     BUFFER+1(PC),A0
0201A6  43FA 001A               LEA     READPA(PC),A1
0201AA  4C91 000F               MOVEM.W (A1),D0-D3
0201AE  4EB9 00001486           JSR     @READ
0201B4  41FA 00EC               LEA     BUFFER(PC),A0
0201B8  1084                    MOVE.B  D4,(A0)
0201BA  6604                    BNE.S   EXEING
0201BC  4FEF 0004               LEA     4(A7),A7
0201C0  4E75            EXEING: RTS
0201C2  00110002        READPA: DC.L    $00110002
0201C6  00800020                DC.L    $00800020
0201CA
0201CA  43FA 013E       VERGL:  LEA     SPEICH(PC),A1
0201CE  45D1            VERGL2: LEA     (A1),A2
0201D0  41FA 00D0               LEA     BUFFER(PC),A0
0201D4  1410                    MOVE.B  (A0),D2
0201D6  1018                    MOVE.B  (A0)+,D0
0201D8  1612                    MOVE.B  (A2),D3
0201DA  121A                    MOVE.B  (A2)+,D1
0201DC  B200                    CMP.B   D0,D1
0201DE  6502                    BCS.S   VERGL3
0201E0  C141                    EXG     D0,D1
0201E2  4880            VERGL3: EXT.W   D0
0201E4  5340                    SUBQ.W  #1,D0
0201E6  B508            VERGL4: CMPM.B  (A0)+,(A2)+
0201E8  6210                    BHI.S   AUFSCH
0201EA  6604                    BNE.S   VERGL5
0201EC  51C8 FFF8               DBRA    D0,VERGL4
0201F0  4882            VERGL5: EXT.W   D2
0201F2  4883                    EXT.W   D3
0201F4  43F1 3002               LEA     02(A1,D3.W),A1
0201F8  60D4                    BRA.S   VERGL2
```

P.21.5 Platz für neuen Record suchen

```
0201FA  200B            AUFSCH: MOVE.L  A3,D0
0201FC  D082                    ADD.L   D2,D0
0201FE  9089                    SUB.L   A1,D0
020200  5340                    SUBQ.W  #1,D0
020202  45D3                    LEA     (A3),A2
020204  15A2 2002      LAUFSH: MOVE.B  -(A2),02(A2,D2.W)
020208  51C8 FFFA              DBRA    D0,LAUFSH
02020C
02020C  41FA 0094      EINSCH: LEA     BUFFER(PC),A0
020210  3002                    MOVE.W  D2,D0
020212  5240                    ADDQ.W  #1,D0
020214  12D8          LEINSH: MOVE.B  (A0)+,(A1)+
020216  51C8 FFFC              DBRA    D0,LEINSH
02021A  47F3 2002              LEA     02(A3,D2.W),A3
02021E  288B                    MOVE.L  A3,(A4)
020220  6000 FF80              BRA     EINGEB
020224
020224                  * ================================ *
020224                  * OPTION 3
020224
020224  7E08            SAVEN:  MOVEQ   #8,D7
020226  4E42                    TRAP    #2
020228  223C 00020306            MOVE.L  #PNTEND,D1
02022E  2081                    MOVE.L  D1,(A0)
020230
020230  7E09                    MOVEQ   #9,D7
020232  4E42                    TRAP    #2
020234  10BC 0001              MOVE.B  #1,(A0)
020238
020238  203A 00CC              MOVE.L  PNTEND(PC),D0
02023C  7E15                    MOVEQ   #21,D7
02023E  4E42                    TRAP    #2
020240
020240  3E3C 000E              MOVE    #14,D7
020244  4E42                    TRAP    #2
020246
020246  4E75                    RTS
```

P.21.6 Aufschieben, Einfügen, Saven

21.6 Die Aufzeichnung auf Diskette

Es kann und soll hier nicht auf die Diskettentechnik eingegangen werden, und die Feinstruktur der benötigten Routinen (Lesen, Schreiben, Klappe zu?, Platz frei? usw.) ist Sache des Diskettenbetriebssystems (Disk Operating System, DOS). Daß hier ein Speicherbereich mit einer nur 34 Byte langen Routine aufgezeichnet werden kann, spricht a) für die Güte des verwendeten (Systema-)DOS und b) für die Güte der mitgelieferten Dokumentation. (Wenn doch so mancher Computerhersteller in diesen Dingen von seiner Geheimniskrämerei abließe!)

In der Routine SAVEN werden wieder TRAP-Befehle benutzt, von denen auch hier nur gesagt werden kann, daß sie ins Innerste des Betriebssystems führen und dort im allgemeinen zu einer Funktionsauswahltabelle ähnlich OPTADR (vorn in diesem Programm). Die Nummer der gewählten Funktion wird bei Systema-DOS in D7.W übergeben. Im einzelnen:

D7 = 8. Mit dieser Funktionsauswahl lädt der Befehl TRAP #2 die Adresse des Ladestartmerkers nach A0. Es handelt sich dabei offenbar um einen Speicherbereich (ähnlich PNTEND), in dem die Anfangsadresse des abzuspeichernden Textes für das Weitere bereitgestellt werden muß. Als erstes ist hier die Adresse von PNTEND aufzuzeichnen, denn sie muß beim Laden von der Diskette richtig wiederhergestellt werden. Es ist wichtig, sie nicht nur nach (A0) in den Ladestartmerker zu kopieren, sondern auch zur späteren Berechnung des auf der Diskette benötigten Platzes in D1 aufzubewahren.

Mit D7 = 9 wird die Adresse des Datenkennungsmerkers nach A0 geholt. Kennung einer Textdatei ist 1. Dementsprechend wird hier diese Kennziffer nach (A0) transferiert.

D7 = 21 wählt eine kompliziertere Funktion aus, deren Ergebnis man vom Inhaltsverzeichnis einer Diskette (directory) her kennt. Zur Berechnung der Dateilänge ist die Angabe der Text-Endadresse in D0 nötig.

Mit D7 = 14 wird schließlich die eigentliche Aufzeichnung ausgelöst. Sie beginnt mit dem Systema-spezifischen Dialog.

A.21.7: Schreiben Sie ein kurzes Programm, mit dem Sie die Adressen des Ladestartmerkers und des Textendemerkers sowie deren Inhalte ermitteln können.

21.7 Die übrigen Optionen

LADEN: Mit D7 = 8 und TRAP #2 wird wieder die Adresse des Ladestartmerkers nach A0 geladen. Dort – also in (A0) – muß die Adresse stehen, von der ab der Text von der Diskette her eingeladen werden soll. Da die Diskettenaufzeichnung mit dem derzeitigen Inhalt des NXTFRE-Pointers begann, müssen die eingelesenen Daten von PNTEND ab im Speicher abgelegt werden. Der Ladevorgang wird durch TRAP #2 in Verbindung mit der Anwahl der Funktion #16 ausgelöst. Auch hier bringt Systema den seinen Usern bekannten Dialog auf den Bildschirm; Einzelheiten hierüber werden die Leser, die nicht mit Systema-DOS arbeiten, kaum interessieren.

Option 5 (Anzeige des Speicherinhalts) ist von den Programmen in Kapitel 20 her hinreichend bekannt.

Option 6 (DRUCKEN) leitet durch Aufruf von @LST die Ausgabe statt zum Bildschirm zum Drucker. Mit @CRT wird sie anschließend wieder zum Bildschirm geschickt.

Option 7 führt nicht zur vorherigen Programmebene zurück, sondern eine Ebene höher zum Grundprogramm, weswegen ein Stapeleintrag ignoriert wird.

Man kann nun darüber philosophieren, ob ein Programm so gut wie möglich oder so gut wie nötig sein sollte. Die Programme in Kapitel 21 sind praktisch brauchbar, denn mit ihnen wurde z.B. das Stichwortverzeichnis erstellt. Sie dürften so transparent sein, daß interessierte Leser sie ihren Bedürfnissen anpassen können. Damit wäre das Ziel erreicht.

```
020248  7E08              LADEN:  MOVEQ   #8,D7
02024A  4E42                      TRAP    #2
02024C  20BC 00020306             MOVE.L  #PNTEND,(A0)
020252  7E10                      MOVEQ   #16,D7
020254  4E42                      TRAP    #2
020256  4E75                      RTS
020258
020258                    * ================================ *
020258                    * OPTION 5
020258
020258  5897              AUSGEB: ADDQ.L  #4,(A7)
02025A  4EB9 00003FEC             JSR     @CLRSCREEN
020260  43FA 00A8         AUSGB2: LEA     SPEICH(PC),A1
020264  1419              AUSGB3: MOVE.B  (A1)+,D2
020266  4882                      EXT.W   D2
020268  41D1                      LEA     (A1),A0
02026A  0C10 00FF                 CMPI.B  #$FF,(A0)
02026E  6718                      BEQ.S   EXAUSG
020270  1018              NXTCHR: MOVE.B  (A0)+,D0
020272  6708                      BEQ.S   NEWLIN
020274  4EB9 00000AE0             JSR     @CO2
02027A  60F4                      BRA.S   NXTCHR
02027C  4EB9 00001698     NEWLIN: JSR     @CRLF
020282  43F1 2001                LEA     01(A1,D2.W),A1
020286  60DC                      BRA.S   AUSGB3
020288  4E75              EXAUSG: RTS
02028A
02028A                    * ================================ *
02028A                    * OPTION 6
02028A
02028A  4EB9 000031A8     DRUKEN: JSR     @LST
020290  6100 FFCE                BSR     AUSGB2
020294  4EB9 000031A0             JSR     @CRT
02029A  4E75                      RTS
02029C
02029C                    * ================================ *
02029C                    * OPTION 7
02029C
02029C  4FEF 0004         GRUPRO: LEA     4(A7),A7
0202A0  4E75                      RTS
0202A2
0202A2                    * ================================ *
0202A2
0202A2                    BUFFER: DS.B    100
020306                    PNTEND: DS.L    1
02030A                    SPEICH: DS.B    3404
```

22
Fließkommazahlen (I)

Um den Erdumfang (ca. 40000000 m) in einem Festkommarechenwerk erfassen zu können, müßte dieses über acht Vorkommastellen verfügen. Der Durchmesser eines Atoms (ca. 0,0000000001 m) erfordert zehn, für hinreichende Genauigkeit sogar dreizehn oder fünfzehn Stellen. Ein (dezimales) Festkommarechenwerk, das beide Bereiche abdeckt, müßte 23 Stellen breit sein. (Das RW der Olivetti P101 (1970) umfaßte fünfundzwanzig Stellen, von denen sich vor Programmablauf bis zu 15 NKS abtrennen ließen.) In den angeführten Beispielen würde ein Festkommarechenwerk für 16 Nullen Speicherplatz (und Rechenzeit) verschenken.

Das Problem ist seit langem durch die «wissenschaftliche Zahldarstellung», die z.B. als 4E7 oder 123.4E-56 von vielen Programmiersprachen übernommen wurde, gelöst. Ihr Pendant auf der Prozessorebene ist das Fließkommarechenwerk.

Drei Fachausdrücke kommen immer wieder vor: Der «Exponent» zur Basis 2 bestimmt die Größenordnung der dargestellten Zahl, die «Mantisse» letztlich die Ziffernfolge, FP = «Floating Point».

22.1 Fließkommazahlen beim QL

Dickens u.a. führen Umwandlungsroutinen FTOD und DTOF (decimal to floating point u.a.) an, bei Pennell (s. Literaturverzeichnis) findet sich eine globale Codierungsbeschreibung. Letztlich half nur der eigene Spürsinn weiter:

Zieht man den Hexdump eines geeigneten SuperBASIC-Programms, findet man schnell die Codierung heraus, die schon während der Programmeingabe errechnet wird:

```
 7      wird zu   F803 7000   0000
15      wird zu   F804 7800   0000
```

31	wird zu	F805 7C00 0000
63	wird zu	F806 7E00 0000
127	wird zu	F807 7F00 0000
4	wird zu	F803 4000 0000
8	wird zu	F804 4000 0000
12.5	wird zu	F804 6400 0000
12.75	wird zu	F804 6600 0000
6.375	wird zu	F803 6600 0000
3.1875	wird zu	F802 6600 0000

A.22.1: Bitte versuchen Sie eine erste Deutung:
a) Wie sind die sechs Bytes der FP-Darstellung aufgegliedert?
b) Wo sind Exponent und Mantisse zu suchen?
c) Welche Gesetzmäßigkeitsparallele besteht zwischen 7, 15, 31, 63 ... und 7000, 7800, 7C00 ...?

Schon im nächsten Abschnitt dieses Kapitels wird zu sehen sein, daß verschiedene Hersteller bzw. verschiedene Softwarehäuser bei ihren FP-Codierungen von unterschiedlichen Normen ausgehen. Nicht immer geht wie hier in der Codierung der Exponent voran. Das höchstwertige Nibble F des .W-Exponenten ist möglicherweise ein Arbeitshinweis für den BASIC-Interpreter. Das folgende Nibble 8 wird von PENNELL als (willkürlicher) Offset bezeichnet. Wir erkennen am LSB des Exponenten, daß dieser – ungenau, aber vereinfacht – bei einer Verdopplung der dargestellten Zahl von 4 auf 8 oder von 7 auf 15, auf 31 um eins erhöht wird; mit anderen Worten: Basis der Zahldarstellung ist die 2.

Auf den Exponent folgt die Mantisse als Langwort. Auch hier wird die Basis 2 deutlich, wenn man Zahl und zugehörige Mantisse binär notiert:

```
 7 = 0000 0111 Mantisse: 0111 0000 0000 0000 ...
15 = 0000 1111 Mantisse: 0111 1000 0000 0000 ...
31 = 0001 1111 Mantisse: 0111 1100 0000 0000 ...
63 = 0011 1111 Mantisse: 0111 1110 0000 0000 ...
```

Es fällt auf, daß das höchstwertige Bit der Mantisse in diesen Beispielen nicht gesetzt ist. Das ist sicherlich kein Zufall; es kennzeichnet üblicherweise eine negative Zahl. Wir gehen den Eigenarten der FP-Codierung beim QL nicht weiter nach, sondern wollen im übernächsten Abschnitt eine eigene Norm entwickeln.

22.2 Fließkommazahlen beim Amiga

Der erste Versuch führt aufs Glatteis. Das Amiga-BASIC benötigt keine Zeilennummern, die das Herumgucken im Speicher erheblich erleichtern. Außerdem erwartet der mit den Grundzügen des M68000 Vertraute ein durchgängiges Wortformat – und staunt. Bild 22.1 zeigt einen fortlaufenden, zeilenweise geordneten Speicherauszug, der der Reihe nach die Zeilen (!)

```
a=12.5
b=12.75
c=12.875
d=12.9375
e=12.96875
f=28.5
g=28.75
h=28.875
i=123.45
j=12345.6
k=-12345.6
```

wiedergibt.

```
2EE0E  0D 00 01 00 00 EA     1D 41 48 00 00 00 00
2EE1B  0D 00 01 00 01 EA     1D 41 4C 00 00 00 00
2EE28  0D 00 01 00 02 EA     1D 41 4E 00 00 00 00
2EE35  0D 00 01 00 03 EA     1D 41 4F 00 00 00 00
2EE42  0D 00 01 00 04 EA     1D 41 80 00 00 00 00
2EE4F  0D 00 01 00 05 EA     1D 41 E4 00 00 00 00
2EE5C  0D 00 01 00 06 EA     1D 41 E6 00 00 00 00
2EE69  0D 00 01 00 07 EA     1D 41 E7 00 00 00 00
2EE76  0D 00 01 00 08 EA     1D 42 F6 E6 66 00 00
2EE83  0D 00 01 00 09 EA     1D 46 40 E6 66 00 00
2EE90  0E 00 01 00 0A EA ED  1D 46 40 E6 66 00 00
```

Bild 22.1 FP-Darstellung beim Amiga (Speicherauszug je BASIC-Zeile)

Die Lücke in den ersten zehn Zeilen des Speicherauszugs existiert in Wahrheit nicht. Hier wurde lediglich willkürlich eine Schreibposition freigelassen, um die Gesetzmäßigkeit hervorzuheben. Die Programmzeilen beginnen mit

einer Längenangabe (vgl. die Programme in Kapitel 20 und 21): Die ersten zehn Zeilen sind tatsächlich 13 Bytes lang, die elfte hat deren 14. Eine zu Worten gruppierte Darstellung hätte hier also in die Irre geführt! Zusätzlich zu allem Vorhergehenden steht in der letzten Zeile ein ED, offenbar eine Amiga-spezifische Kennzeichnung negativer Zahlen. In der fünften Zahlenkolonne wird ganz einfach nur gezählt; möglicherweise sind die im Programm vorkommenden Variablennamen in einer Tabelle verzeichnet, so daß nicht bei jedem Durchlauf mühsam die Namen verglichen werden müssen. Wahrscheinlich werden interpreterintern die Variablen einfach durch ihre Ordnungszahl in dieser Tabelle – und parallel dazu vielleicht in einem Zahlenstapel – identifiziert. Die fröhliche Schnelligkeit des Amiga-BASIC fände so eine (von sicherlich vielen) Teilerklärung(en). Bild 22.2 stellt die ersten zehn Zahlen etwas übersichtlicher dar.

```
    12.5       wird zu    .. 41 48000000
    12.75      wird zu    .. 41 4C000000
    12.875     wird zu    .. 41 4E000000
    12.9375    wird zu    .. 41 4F000000
    12.96875   wird zu    .. 41 4F800000
    28.5       wird zu    .. 41 E4000000
    28.75      wird zu    .. 41 E6000000
    28.875     wird zu    .. 41 E7000000
   123.45      wird zu    .. 42 F6E66600
 12345.6       wird zu    .. 46 40E66600
```

Bild 22.2 FP-Darstellung beim Amiga (den Zahlen zugeordnet)

Amiga-BASIC kennt vier numerische Zahlentypen:

1. Ganzzahlvariable «kurz», Kennzeichen: «%»
2. Ganzzahlvariable «lang», Kennzeichen: «&»
3. Realzahlvariable einfacher Genauigkeit, Kennzeichen: keins oder «!»
4. Realzahlvariable doppelter Genauigkeit, Kennzeichen: «#»

Es wurde absichtlich nur ein Zahlentyp herausgegriffen, und auch an ihm soll nur Grundsätzliches betrachtet werden.

```
Dez'zahl      Binärzahl                 Mantisse
   12.5       = 0000 1100.1000 0000     0100 1000 0000 0000
   12.75      = 0000 1100.1100 0000     0100 1100 0000 0000
   12.875     = 0000 1100.1110 0000     0100 1110 0000 0000
   12.9375    = 0000 1100.1111 0000     0100 1111 0000 0000
   12.96875   = 0000 1100.1111 1000     0100 1111 1000 0000
   28.5       = 0001 1100.1000 0000     1110 0100 0000 0000
   28.75      = 0001 1100.1100 0000     1110 0110 0000 0000
   28.875     = 0001 1100.1110 0000     1110 0111 0000 0000
  123.45      = 0111 1011.0111 0011..   1111 0110 1110 0110 ....
12345.6
  = 0011 0000 0011 1001.1001 1001..     0100 0000 1110 0110 ....
```

Bild 22.3 Die Beispielzahlen in binärer und in FP-Darstellung

Den Dezimalbrüchen entsprechen rechnerintern Binärbrüche. Die Stellenwerte hinter dem Binärpunkt sind Halbe, Viertel, Achtel usw. Die Dezimalzahl 12,75 sieht also als reelle Binärzahl so aus: 1100.11, was bedeutet: je ein Achter, Vierer, Halbes, Viertel. Noch eine Übersicht bringt Bild 22.3.

Auch hier fällt – wie beim QL – auf, daß sich die Bitmuster der Binärzahl in der Mantisse (abgesehen vom evtl. Überschreiten der Stellenkapazität) unmittelbar in der Mantisse wiederfinden, allerdings um einige Stellen verschoben. Während beim QL das MSb der Mantisse bei positiven Zahlen gelöscht war, ist es hier manchmal gesetzt: Bei der Steigerung von 12.xx nach 28.xx wurde eine Zweierpotenz überschritten. Hier wird in diesem Fall nicht der Exponent um eins erhöht, sondern das MSb mitbenutzt; ein einfacher Überschlag erhellt den Hintergrund:

$$\frac{12345{,}6}{12{,}5} \approx 988 \approx 2^{10}$$

Wie Bild 22.2 zeigt, wird in diesem Fall der Exponent nicht um 10 erhöht, sondern nur um 5, was bedeutet, daß der Exponent nicht zur Basis 2, sondern zur Basis 4 steht: Interessierte Leser werden spätestens im nächsten Kapitel erkennen, warum auch das Zeit sparen hilft.

Wieso Exponent zur Basis 4, wo doch $4^{\$41} = 4^{65} \approx 1.36E39$, während lediglich 12.xx dargestellt wird? Lassen Sie sich vom Offset $40 nicht täuschen; wir kommen bald darauf zurück. Eine Feindeutung würde hier zu weit führen.

22.3 Unsere Fließkomma-Norm

Eine erschöpfende Darstellung der Problematik würde den Rahmen sprengen und am Thema vorbeigehen. Wir beschränken uns daher auf die Grundstrukturen und verzichten weithin auf die negativen Zahlen. Unsere Rechenbasis soll wie weitverbreitet die 2 sein. Dem Exponent wird Wortbreite zugestanden, obgleich sie sicherlich nicht ausgeschöpft werden wird, der Mantisse Langwortformat.

Wir planen ein FP-Rechenwerk mit Stapelstruktur, etwa so wie bei FORTH gebräuchlich. Angenommener Stapelzustand vor und nach dem Befehl «+»:

Vorher		Danach
111	← top of stack →	777
666		555
555		???

Die beiden oberen Zahlen werden dem Befehl entsprechend verknüpft. Das Resultat steht im TOS (top of stack), alle übrigen Zahlen rücken entsprechend auf.

Der Exponent bleibt ohne Offset, das MSb der Mantisse repräsentiert das Vorzeichen. Man kann sich außerdem an seiner Stelle den Binärpunkt vorstellen, so daß die Mantisse 0111 ein Halbes, ein Viertel und ein Achtel darstellen würde. Das führt konkret zu folgender Codierung:

$$\begin{aligned} 12{,}75 = 12 + \frac{1}{2} + \frac{1}{4} &= 1100.1100\ 0000\ z\ 0 \\ &= 0110.0110\ 0000\ z\ 1 \\ &= 0011.0011\ 0000\ z\ 2 \\ &= 0001.1001\ 1000\ z\ 3 \\ &= 0000.1100\ 1100\ z\ 4 \\ &= 0000.0110\ 0110\ z\ 5 \end{aligned}$$

Da aber das höchstwertige Bit der Mantisse für die Codierung des Vorzeichens frei bleiben soll, beginnen ihre signifikanten Stellen im nächstfolgenden, woraus schließlich folgt:

$$FP(12.75) = 66z4 \text{ bzw. } 0004\ 66000000$$

22.4 Vom Zahlenstapelkonzept

Inspiriert von UPN (s. Glossar) und der Programmiersprache FORTH und um bei künftigen Experimenten beweglicher zu sein, ordnen wir die zum Rechnen bereitgestellten Zahlen in einem Stapel an. Ein Programm zur Berechnung von 12,75 + 6,5 sieht dann so aus:

Eingeben	12.75
Eingeben	6.5
Addieren	
Ausgeben	(19.25)

Im Stapel spielt sich dabei folgendes ab:

Urzustand	1. Eingabe	2. Eingabe	Addition
xx xxxxxx	04 660000	03 680000	05 4D0000
yy yyyyyy	xx xxxxxx	04 660000	xx xxxxxx
zz zzzzzz	yy yyyyyy	xx xxxxxx	yy yyyyyy

Das Aufrücken der Zahlen im Stapel bei der Eingabe einer neuen läßt sich auch so deuten: Zunächst wird die erste Zahl dupliziert (FORTH-Befehl DUP). Alle Zahlen rücken um einen Platz (bildlich gesprochen) nach unten. Da die Zahlen ja in Wahrheit nur von einer Position zur anderen kopiert werden, ist danach die oberste zweimal vorhanden. Die Eingabe einer neuen Zahl beginnt mit dieser DUPlizierung. (DUP kann aber auch – etwa um eine Zahl zu quadrieren – einzeln aufgerufen werden.) Danach erfolgt die eigentliche Eingabe.

DROP bewirkt das Gegenteil von DUP: Die oberste Zahl wird verworfen; die anderen rücken auf. Auch eine ausgegebene Zahl verschwindet vom Stapel; die Ausgabe endet mit DROP. Soll eine Zahl trotz Ausgabe erhalten bleiben, muß sie vorher DUPliziert werden.

22.5 Die Konvertierung DEZ → FP

P.22.1 beginnt mit einem kurzen Experimentierprogramm. Das anschließende ORG soll das spätere Experimentieren erleichtern. Der Quelltext wird einmal assembliert, und dann steht das Maschinenprogramm einsatzbereit im Speicher. (Man kann es auch für sich allein auf Diskette aufzeichnen.) Dann mag man separat die verschiedensten Experimentierprogramme ent-

```
020000  6100 2FFE       EXPROG: BSR     EINGEB
020004  6100 3086               BSR     AUSGEB
020008  60F6                    BRA.S   EXPROG
02000A
023000                          ORG     $23000
023000
023000  6100 00F6       EINGEB: BSR     DUP
023004  41FA 0128               LEA     READPA(PC),A0
023008  4C90 000F               MOVEM.W (A0),D0-D3
02300C  41FA 0134               LEA     EBER(PC),A0
023010  4EB8 1486               JSR     @READ.S
023014
023014  41FA 012C       SUCHPU: LEA     EBER(PC),A0
023018  43D0                    LEA     (A0),A1
02301A  0C19 002E       LOSUPU: CMPI.B  #'.',(A1)+
02301E  57CC FFFA               DBEQ    D4,LOSUPU
023022  4229 FFFF               CLR.B   -1(A1)
023026  4EB8 172C               JSR     @WERT.S
02302A  2800                    MOVE.L  D0,D4
02302C  41D1                    LEA     (A1),A0
02302E  4EB8 172C               JSR     @WERT.S
023032  323C 1388               MOVE.W  #5000,D1
023036  48C1                    EXT.L   D1
023038  741F                    MOVEQ   #$1F,D2
02303A  4A80            SETZEX: TST.L   D0
02303C  6714                    BEQ.S   LSMANT
02303E  4A81                    TST.L   D1
023040  6710                    BEQ.S   LSMANT
023042  5342                    SUBQ.W  #1,D2
023044  D884                    ADD.L   D4,D4
023046  B081                    CMP.L   D1,D0
023048  6504                    BCS.S   SETNUL
02304A  9081                    SUB.L   D1,D0
02304C  5284                    ADDQ.L  #1,D4
02304E  E289            SETNUL: LSR.L   #1,D1
023050  60E8                    BRA.S   SETZEX
```

P.22.1 Konvertierung DEZ → FP (I)

wickeln und muß ihnen nur die benötigten Einsprungadressen entweder mit EQU oder in einer Tabelle mitteilen.

Für die Eingabe gilt hier die Norm, daß jede Zahl mit «.» und vier Nachkommastellen enden muß. 3 ist also einzugeben als 3.0000. Wie bekannt, steht nach der Eingabe in D4 die Anzahl der gültigen Zeichen.

SUCHPU sucht mittels (A1)+ und Schleifenzähler D4 den Dezimalpunkt. Um die wesentlichen Konturen herauszuarbeiten, wurde auf jegliche Fehlerkontrolle verzichtet; wir verlassen uns auf korrekte Eingaben.

Wurde der Punkt gefunden, wird er durch das Schlußzeichen 00 ersetzt. Nach Eingabe von 12.7500 sieht jetzt der Eingabebereich so aus: 31 32 00 37 35 30 30 00. A0 zeigt auf EBER-Anfang, so daß mit @WERT der String sofort in eine Hexadezimalzahl umgewandelt werden kann, die in D0 aufgebaut wird.

Nun übernimmt A0 den Inhalt von A1 und ist also auf die erste Nachkommastelle gerichtet. D0 = 7500 = ($)00001D4C. Jetzt brauchte nur noch durch 10000 = $2710 geteilt zu werden, um 0.75 zu erhalten – natürlich unter Linksverschiebung, so daß sich intern alles vor dem Komma abspielt. Durch eine Variante der bitweisen Division kann man Überlaufprobleme umgehen. Dividiert man dabei statt durch 10000 nur durch 5000, erhält man sofort gültige Nachkommabits.

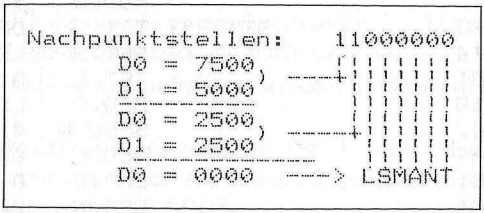


Bild 22.4 Bitweise Division 7500 : 5000

Bild 22.4 läßt die mit den Halben beginnende richtige Bitfolge für .75 erkennen. Wenn der Dividend oder der Teilersubtrahend zu 0 wurde, ist eine Fortsetzung sinnlos. Bis zu dieser Abbruchbedingung ist also folgende Prozedur auszuführen:

Prüfe Abbruchbedingung in D0 ((Rest der) Nachkommastellen);
prüfe Abbruchbedingung in D1 (Teilersubtrahend);
shifte Inhalt von D4 (Vorkommastellen) um eine Binärstelle nach links;
ist der Inhalt von D0 (Nachkommastellen) kleiner als der Teilersubtrahend

in D1, lasse das LSb von D4 auf Null; anderenfalls addiere eine 1 und vermindere den Inhalt von D0 um den Teilersubtrahend.

Diese Schleife wurde SETze ZweierEXponent genannt, weil in ihr der Zweierexponent in D2 hergeleitet wird. Unter Umgehung der Mathematik wurde D2 eingangs mit dem Erfahrungswert 31 geladen. Da jedes Linksshiften den Zahlenwert der Mantisse verdoppelt, muß im gleichen Schritt der Exponent herabgezählt werden; dazu SUBQ.W #1,D2.

Die Mantisse hat vereinbarungsgemäß .L-Format. Ihr Bit 31 soll das Vorzeichen repräsentieren; das höchstwertige numerisch zu deutende Bit ist #30; in unserer «normierten FP-Darstellung» ist es gesetzt. Da das i.a. nicht zutrifft, wird der Inhalt von D4 (Mantisse) so lange unter gleichzeitigem Dekrementieren des Exponenten nach links geshiftet (hier durch ADD.L D4,D4), bis der BitTeST ergibt «Bit gesetzt», das heißt Not Equal. Nun werden bei FERTIG Exponent und Mantisse auf dem Zahlenstapel abgelegt.

Später verzichtbar, wird bis auf weiteres die interne FP-Darstellung der eingegebenen Zahl angezeigt.

```
023052  0804 001E       LSMANT: BTST     #30,D4
023056  6606                    BNE.S    FERTIG
023058  D884                    ADD.L    D4,D4
02305A  5342                    SUBQ.W   #1,D2
02305C  60F4                    BRA.S    LSMANT
02305E
02305E  45FA 00FE       FERTIG: LEA      ZALSTA(PC),A2
023062  34C2                    MOVE.W   D2,(A2)+
023064  24C4                    MOVE.L   D4,(A2)+
023066
023066  3002             AUFEXP: MOVE.W   D2,D0
023068  41FA 00E4                LEA      ABER(PC),A0
02306C  4EB8 1660                JSR      @PRINT4X.S
023070  10FC 0020                MOVE.B   #' ',(A0)+
023074  2004             AUFMAN: MOVE.L   D4,D0
023076  4EB8 164A                JSR      @PRINT8X.S
02307A  41FA 00D2                LEA      ABER(PC),A0
02307E  43FA 00B6                LEA      WRITP1(PC),A1
023082  4C91 0007                MOVEM.W  (A1),D0-D2
023086  4EB8 1386                JSR      @WRITE.S
02308A  4E75                    RTS
```

P.22.1 Konvertierung DEZ → FP (II)

22.6 Die Konvertierung FP → DEZ

Die im TOS (top of stack) eingetragene Zahl wird zur Ausgabe in eine Dezimalzahl umgewandelt. Zuerst werden Exponent und Mantisse nach D4 bzw. D3 geholt.

```
02308C  45FA 00D0        AUSGEB: LEA     ZALSTA(PC),A2
023090  381A                    MOVE.W  (A2)+,D4
023092  261A                    MOVE.L  (A2)+,D3
023094  7000                    MOVEQ   #0,D0
023096  E383            TRENVK: ASL.L   #1,D3
023098  E390                    ROXL.L  #1,D0
02309A  51CC FFFA               DBRA    D4,TRENVK
02309E  41FA 00AE               LEA     ABER(PC),A0
0230A2  4EB8 15FC               JSR     @PRINT4D.S
0230A6  10FC 002E               MOVE.B  #'.',(A0)+
0230AA  10BC 0000               MOVE.B  #0,(A0)
0230AE  47D0                    LEA     (A0),A3
0230B0  2003                    MOVE.L  D3,D0
0230B2  EC88                    LSR.L   #6,D0
0230B4  EE88                    LSR.L   #7,D0
0230B6  612E                    BSR.S   NKSTEL
0230B8  612C                    BSR.S   NKSTEL
0230BA  612A                    BSR.S   NKSTEL
0230BC  6128                    BSR.S   NKSTEL
0230BE  10FC 0020               MOVE.B  #' ',(A0)+
0230C2  10FC 0020               MOVE.B  #' ',(A0)+
0230C6  10FC 0020               MOVE.B  #' ',(A0)+
0230CA  10FC 0000               MOVE.B  #0,(A0)+
0230CE  45FA 006C               LEA     WRITP2(PC),A2
0230D2  4C92 0007               MOVEM.W (A2),D0-D2
0230D6  41FA 0076               LEA     ABER(PC),A0
0230DA  4EB8 1386               JSR     @WRITE.S
0230DE  6138                    BSR.S   KEYIN
0230E0  6100 0026               BSR     DROP
0230E4  4E75                    RTS
0230E6
0230E6  80FC CCCD        NKSTEL: DIVU    #52429,D0
0230EA  1080                    MOVE.B  D0,(A0)
0230EC  0618 0030               ADD.B   #$30,(A0)+
0230F0  4840                    SWAP    D0
0230F2  C0FC 000A               MULU    #10,D0
0230F6  4E75                    RTS
```

P.22.2 Konvertierung FP → DEZ

```
                        D0        X      D3
Vorher:              01110000    ?   10011100
ASL.L   #1,D3                    1   00111000
ROXL.L  #1,D0        11100001    0
ASL.L   #1,D3                    0   01110000
ROXL.L  #1,d0        11000010    1
```

Bild 22.5 Das Hinübershiften von D3 nach D0

Wir gehen nun den umgekehrten Weg, lesen die Mantisse binär und trennen beim MSb angefangen eine Stelle mehr, als der Exponent angibt, als Vorkommastellen ab; sie werden in eine Dezimalzahl umgewandelt. Die Schleife TRENVK trennt diese künftigen Vorkommastellen ab; sie werden über das X-Flag von D3 nach D0 geshiftet. D4 fungiert dabei unmittelbar als Schleifenzähler (Bild 22.5).

@PRINT4D bereitet nun den Inhalt von D0 zum String im Ausgabebereich auf. Das Schlußzeichen 00 wird sofort von «.» überschrieben und vorsichtshalber – auch im Hinblick auf späteren Programmausbau – eine Null dahintergesetzt, deren Position in A3 zwischengespeichert wird.

Die Aufbereitung der Nachkommastellen mit einer @PRINTxD-Routine liegt nahe, führt aber zu Falschausgaben, weil führende Nullen unterdrückt werden: 44.0625 würde zu 44.625 werden! Statt dessen werden vier Nachkommastellen durch Division durch Zehntel, Hundertstel usw. zurückgewonnen, was durch viermaligen Aufruf der Subroutine NKSTEL geschieht – leider aber ein wenig undurchsichtig wegen der FP-Darstellung:

Vor dem ersten Aufruf wird das MSb des verbliebenen Mantissenrestes um 13 Binärstellen nach rechts geshiftet, um Überläufe bei den nachfolgenden Divisionen zu vermeiden. Später wird dann entsprechend zusätzlich mit 2^{13} multipliziert. Weil der M68000 aber keine Nachkommastellen kennt, müssen wir das Ganze wiederum durch eine Quasi-Erweiterung vor das Komma verlegen. Machen wir's konkret: Die Restmantisse möge den Gegenwert von .75 enthalten. Ein Halbes und ein Viertel sah zunächst so aus: C0000000. Nach dem Shiften wurde daraus 00060000, was dezimal 393216 entspricht, woraus hinter dem Komma 7500 werden sollen. 393216 : 7500 = 52.4288. Da aber während der Rechnung dreimal mit 10 multipliziert wird, muß letztlich statt durch Zehntel durch 52428.8 geteilt werden. Die nun verblie-

```
00060000

1.NKSTEL    2.NKSTEL    3.NKSTEL    4.NKSTEL

66650007    CCBE0004    CC370009    C6F10009
00076665    0004CCBE    0009CC37    0009C6F1
0003FFF2    0007FF6C    0007FA26    0007C56A

2E37xxxx    2E3734xx    2E373439    2E37343939
  . 7         . 7 4       . 7 4 9     . 7 4 9 9
```

Bild 22.6 Die Vorgänge in D0 und im ABER bei den vier Aufrufen von NKSTEL

benen Nachkommastellen runden wir auf und nehmen einen minimalen Fehler in Kauf, indem wir durch 52429 teilen. Auch hier wird die Anschaulichkeit vielen Lesern willkommen sein (Bild 22.6).

Zunächst zum Befehl DIVU (divide unsigned, vorzeichenlose Division): DIVU D1,D2 dividiert das in D2 stehende Langwort durch das Wort in D1. Der ganzzahlige Wortquotient steht anschließend im LSW von D2, der Divisionsrest im MSW von D2. Unser Beispiel

$$\$00060000 : \$CCCD = \$0007 \text{ Rest } \$6665$$

unterscheidet sich von dieser Erklärung nur dadurch, daß der Quelloperand unmittelbar gegeben wurde.

Bitte vergleichen Sie das mit dem Listing und mit Bild 22.6; Sie werden erkennen, daß dieses Ergebnis in D0 zu finden ist:

$$D0 = 66650007$$

Nach dem SWAPpen: $D0 = 00076665$

Die nachfolgende Verzehnfachung ignoriert das MSW; ihr Resultat ist $10 * \$6665 = \$0003FFF2$. Sie können den weiteren Fortgang unmittelbar aus Bild 22.6 ersehen. – NKSTEL endet mit der Fortschreibung des Strings im Ausgabebereich.

Zur Ausgabe auf dem Bildschirm ist nichts mehr zu sagen, außer daß das Bild erhalten bleibt, bis eine Taste gedrückt wird. Wird «G» gedrückt, geht es zurück ins Grundprogramm.

Nach der Vorstellung des Zahlenstapelkonzepts und nach der Besprechung von Verschiebungen von Speicherbereichen in den Sortierprogrammen werden sich Erklärungen des Programmrestes erübrigen.

```
0230F8  41FA 00A0       DUP:    LEA     ZALSTA+60(PC),A0
0230FC  701D                    MOVEQ   #29,D0
0230FE  3160 0006       DUPLOP: MOVE.W  -(A0),6(A0)
023102  51C8 FFFA              DBRA    D0,DUPLOP
023106  4E75                   RTS
023108
023108  41FA 0054       DROP:   LEA     ZALSTA(PC),A0
02310C  701D                    MOVEQ   #29,D0
02310E  30E8 0006       DROLOP: MOVE.W  6(A0),(A0)+
023112  51C8 FFFA              DBRA    D0,DROLOP
023116  4E75                   RTS
023118
023118  4EB8 09D8       KEYIN:  JSR     @CSTS.S
02311C  67FA                    BEQ.S   KEYIN
02311E  4EB8 0A00               JSR     @CI.S
023122  0C00 0047               CMPI.B  #'G',D0
023126  6604                    BNE.S   EXKEY
023128  4FEF 0008               LEA     8(A7),A7
02312C  4E75            EXKEY:  RTS
02312E
02312E  00110020        READPA: DC.L    $00110020
023132  0020000A                DC.L    $0020000A
023136  00110020        WRITP1: DC.L    $00110020
02313A  0010                    DC.W    $0010
02313C  00110020        WRITP2: DC.L    $00110020
023140  0002                    DC.W    $0002
023142                  EBER:   DS.B    11
02314D  00                      DS.B    0
02314E                  ABER:   DS.L    4
02315E                  ZALSTA: DS.W    60
```

P.22.3 Drei Routinen und der Arbeitsbereich

22.7 Programmablauf, Genauigkeit, Verbesserungsmöglichkeiten

Eingabe	FP-Darstellung	Ausgabe
12.0000	0004 60000000	12.0000
12.1000	0004 60CD4000	12.1002
12.0100	0004 6014C000	12.0101
12.0010	0004 60024000	12.0010
12.0001	0004 60004000	12.0001
765.4323	000A 5FADD600	765.4323
0.9999	0000 7FFC0000	0.9998
0.8002	0000 66700000	0.8002
0.5000	0000 40000000	0.4999
0.3000	FFFF 4CE00000	0.0000

Bild 22.7 Beispiele zur Genauigkeitskontrolle

Bild 22.7 zeigt (in abgewandelter Anordnung) einige konkrete Zahlenbeispiele. Man erkennt dabei einen bedauerlichen, aber doch zu verschmerzenden Konvertierungsfehler. Er ließe sich sicherlich noch etwas verringern, wenn man nach dem letzten NKSTEL-Aufruf noch eine Rundung einbaute, auf die wir jedoch übersichtlichkeitshalber verzichten.

Ein anderer Fehler ist schlimmer: Bei Eingabe von Zahlen $< 0{,}5$ wird der Exponent negativ, was hier zu sinnlosen Ergebnissen führt. Man vermeidet das in der Praxis, indem man (vgl. Abschnitt 22.1) den Exponenten anders normiert, z.B. \$40 = 0 setzt (Offset).

Während bei uns noch vermeidbare Vereinfachungen hinzukamen, liegt die wesentliche Ursache des Konvertierungsfehlers darin, daß der Allerwelts-Dezimalbruch 0,1 in binärer Notierung periodisch ist:

$$0{,}1 = 0.0001100110011\ldots$$

Sie erkennen diese Periodizität sehr schön im NKSTEL-Divisor wieder: \$CCCD = 1100110011001101, wobei sich die gewählte Aufrundung korrekt widerspiegelt. Auch in Bild 22.4 sind (leider vorzeitig abbrechende) Perioden zu erkennen.

22.8 Die Fließkomma-Addition

Den Vereinbarungen von Abschnitt 22.3 und der Beobachtung aus Abschnitt 22.7 entsprechend lassen wir nur noch Zahlen >= 0,5 zur Verarbeitung zu. Solche Zahlen sollen hier addiert werden.

Beispiel 1: 765,4323 + 12,75

Dasselbe in FP-Darstellung:

```
  000A 5FADD600 (= Zahl 1)
+ 0004 66000000 (= Zahl 2)
```

Bekanntlich darf man Produkte mit unterschiedlichen Potenzen (gleicher Basis) nicht ohne weiteres zusammenfassen. Hier bleibt nur die Abhilfe, die Mantisse der zweiten Zahl (kurz M2) um 6 Stellen nach rechts zu shiften, was einer Division durch 64 entspricht, und den Exponenten (kurz E2) entsprechend um 6 zu erhöhen:

```
                 000A 5FADD600
Danach darf    + 000A 01980000
addiert werden:  000A 61145D600
```

Beispiel 2: 12,75 + 7,125

Dasselbe in FP-Darstellung:

```
  0004 66000000
+ 0003 72000000
```

Hieraus wird nach dem erforderlichen Rechtsshiften

```
                 0004 66000000
und es darf    + 0004 39000000
addiert werden:  0004 9F000000
```

Hier hat sich aber aus zwei positiven Zahlen eine nicht nur nach unserer Norm negative Summe eingestellt: Das MSb der Mantisse wurde gesetzt! Wegen der Übereinstimmung mit der Zweierkomplementarithmetik wird hier das V-Flag (Über*lauf*bit) gesetzt. In diesem Falle muß die Summenmantisse unter Erhöhung des Summenexponenten um eine Stelle nach rechts geshiftet werden:

0004 9F000000 = 0005 4F800000 (= 19,875).

```
020000  48E7 F080       ADDIRE: MOVEM.L D0-D3/A0,-(A7)
020004  41FA 0042               LEA     ZALSTA(PC),A0
020008  3018                    MOVE.W  (A0)+,D0
02000A  2218                    MOVE.L  (A0)+,D1
02000C  3418                    MOVE.W  (A0)+,D2
02000E  2618                    MOVE.L  (A0)+,D3
020010  B440               VERGEX: CMP.W   D0,D2
020012  670E                    BEQ.S   ADDIRN
020014  6506                    BCS.S   D3RESH
020016  E281               D1RESH: ASR.L   #1,D1
020018  5240                    ADDQ.W  #1,D0
02001A  60F4                    BRA.S   VERGEX
02001C  E283               D3RESH: ASR.L   #1,D3
02001E  5242                    ADDQ.W  #1,D2
020020  60EE                    BRA.S   VERGEX
020022
020022  6100 0018          ADDIRN: BSR     DROP
020026  D283                    ADD.L   D3,D1
020028  6804                    BVC.S   NOSHIR
02002A  E289                    LSR.L   #1,D1
02002C  5240                    ADDQ.W  #1,D0
02002E  41FA 000E          NOSHIR: LEA     RESULT(PC),A0
020032  30C0                    MOVE.W  D0,(A0)+
020034  20C1                    MOVE.L  D1,(A0)+
020036  4CDF 010F                MOVEM.L (A7)+,D0-D3/A0
02003A  4E75                    RTS
02003C
02003C  4E75               DROP:   RTS
02003E
02003E                     RESULT: DS.W    3
020044  3E3E3E3E           KENNER: DC.L    '>>>>'
020048  0004               ZALSTA: DC.W    $0004
02004A  66000000                   DC.L    $66000000
02004E  0003                       DC.W    $0003
020050  72000000                   DC.L    $72000000
020054                             END
```

P.22.4 FP-Addition zweier Zahlen >= 0,5

Disposition: Die beiden Summanden Z1 und Z2 werden zur Addition vom Zahlenstapel geholt und die Summe auf dem Zahlenstapel abgelegt. Registerbelegung:

D0.W: E1 D1.L: M1 D2.W: E2 D3.L: M2 A0: Stapelzeiger

Um nicht jedesmal vollen Aufwand treiben zu müssen, vereinfachen wir das Teilproblem durch Vorgabe der beiden FP-Summanden mittels DC-Direktive. An das beabsichtigte DROP erinnert der unmittelbar zu einem RTS führende Aufruf. Entsprechend wird die FP-Summe im Arbeitsbereich abgelegt. Das Langwort «>>>>» soll die Orientierung im Speicher erleichtern. Eins wurde hinzugenommen: In längeren Programmen wird man auch mit der beachtlichen Registerzahl des M68000 nicht auskommen. Im Hinblick auf Späteres wurde daher P.22.4 als Routine ausgelegt, die keine übernommenen Registerinhalte verändert. Dazu werden zu Anfang die Inhalte aller beteiligten Register auf dem User-Stapel abgelegt und vor dem RTS von dort zurückgeholt. Der Anwenderstapel und der Zahlenstapel sind zwei völlig verschiedene Speicherbereiche.

Bei VERGEX werden die beiden Exponenten miteinander verglichen. Nur wenn die beiden Exponenten einander gleich sind, geht es weiter zur Addition. Ist E1 (in D0) größer, wird Carry gesetzt und die in D3 stehende M2 bei gleichzeitiger Erhöhung von E2 nach rechts geshiftet; zurück zu einem neuen VERGEX. Im umgekehrten Fall wird D1RESH entsprechend durchlaufen.

Das Addieren beginnt mit dem schon angesprochenen Pseudo-DROP. Die Normierung bei Überlauf dürfte verständlich sein, und auch zur Ablage im RESULTatfeld ist nichts zu bemerken. – Sie täten gut daran, mit unterschiedlichen Zahlvorgaben zu experimentieren. Vielleicht wollen Sie die Routine auch schon in das Konvertierungsprogramm einbauen?

22.9 Die Additionsbefehle

In P.22.4 wurde ohne besondere Abwägung der Befehl ADD eingesetzt. Der folgende Exkurs gibt einen Überblick:

ABCD Dq,Dz und ABCD −(Aq),−(A)

Diese beiden schon in P.18.2 benutzten und in A.18.2 etwas näher untersuchten Befehle addieren binär codierte Dezimalzahlen. Da sie nur .B-Daten verarbeiten, werden sie im allgemeinen aneinandergereiht werden, wozu sich

besonders gut die −(An)-Version eignet. Ein evtl. Übertrag wird dabei im X-Flag übergeben, so daß man dies vor der ersten Addition löschen muß.

Die Flags werden genau ihrer Definition (vgl. Abschnitt 7.1) entsprechend beeinflußt.

ADD.X <ea>,Dn und ADD.X Dn,<ea>

Dieser Befehl addiert die beiden Operanden hexadezimal. Die Flags werden ganz regulär dem Ergebnis angepaßt.

ADDA.X <ea>,An

Ist ein Adreßregister Ziel einer Addition, wird ADD zu ADDA. Für X sind nur die beiden Adreßformate .W und .L zugelassen. Flags werden nicht verändert.

ADDI.X #Konstante,<ea>

Dieser Befehl addiert eine 8-, 16- oder 32-bit-Konstante zum Ziel. Die Flags reagieren regulär.

ADDQ.X #Konstante,<ea>

Hier wird eine 3-bit-Konstante (8, 1 ... 7) zum Ziel addiert. Das geht besonders schnell (quick), weil die Konstante bereits im Opcode enthalten ist. Da eine Addition von 0 wenig sinnvoll wäre, repräsentiert das Tripel 000 den Summanden 8. Die Flags reagieren regulär.

ADDX.X Dq,Dz und ADDX.X −(Aq),−(Az)

Diese Befehle haben, obwohl sie eine hexadezimale Addition bewirken, viel mit ABCD gemein: Das X-Flag wird in die Summierung einbezogen, so daß es vor der ersten Teiladdition in der Regel gelöscht sein muß. Hingegen sollte man aus den gleichen Gründen wie bei ABCD vor der (ersten) Addition das Z-Flag setzen. Ansonsten werden die Flags regulär beeinflußt.

23
Die Flags des CCR (II)

Wir haben in Kapitel 7 schon Grundsätzliches über die Flags gehört; die Beobachtung eines konkreten Überlaufs in P.22.8 regt an, uns mit den Flags genauer zu beschäftigen.

Das eigentliche Maschinenprogramm rettet die benutzten Register auf den Stapel, holt zwei .B-Zahlen aus dem Puffer, addiert sie und legt den Inhalt des Statusregisters im Puffer ab. Da hiervon das Systembyte noch nicht interessiert, wird später nur sein LSB, das CCR, ausgelesen. – Das Ganze wurde, um größtmögliche Flexibilität zu erlangen, in ein BASIC-Programm eingebettet.

23.1 Die Flags und die Addition

Das BASIC-Programm simuliert die 65536 möglichen .B-Additionen. Da aber nach 5 + 0, 5 + 1, 5 + 2 ... (mit Vorsicht) extrapoliert werden darf, werden für die Summanden nur die 16 Zahlen 00, 01, 02, 03, 7C ... 83, FC, FD, FE, FF bereitgestellt. Jede der nun 16 * 16 = 256 möglichen Additionen wird nun durch einen Punkt in einer 16 * 16-Matrix symbolisiert. Wurde aber bei einer Addition Carryflag gesetzt, wird der Punkt durch ein «c» ersetzt. Man bekommt nun auf dem Bildschirm eine grafische Darstellung der Fälle, in denen Carry gesetzt wurde. Um das Ergebnis vorwegzunehmen: Bei Anordnung der vorzeichenlosen Zahlen (00 ... FF) nach Art der Cartesischen Ebene sind in der rechten oberen Fast-Halbebene (oberhalb der Nebendiagonale) c's zu sehen. Die oberste Zeile .ccccccccccccc ist so zu lesen:

$$FF + 00 = FF, \quad FF + 01 = (1)00, \quad FF + 02 = (1)01 \ldots,$$

was bedeutet, daß von der zweiten symbolisierten Addition an stets ein Übertrag stattfindet.

```
REM 000   48E7 E080   MOVEM.L   D0-D2/A0,-(A7)
REM 004   41FA 0010   LEA       SUMMD1(PC),A0
REM 008   1018        MOVE.B    (A0)+,D0
REM 00A   1218        MOVE.B    (A0)+,D1
REM 00C   D001        ADD.B     D1,D0
REM 00E   40D8        MOVE      SR,(A0)+
REM 010   4CDF 0107   MOVEM.L   (A7)+,D0-D2/A0
REM 014   4E75        RTS
REM
REM 016   SUMMD1
REM 017   SUMMD2
REM 018   SR
REM 019   CCR
REM                   END
```

```
GOSUB holzahl
code$=STRING$(30,0): GOSUB holprog
FOR zeile=16 TO 1 STEP -1
  dz$="": POKE anfad+22,z(zeile)
  FOR spalte=1 TO 16
    POKE anfad+23,z(spalte)
    CALL anfad
    z$=".": IF PEEK(anfad+25) AND 1 THEN z$="c"
    dz$=dz$+z$
  NEXT spalte
  PRINT dz$
NEXT zeile
END

holzahl: DIM z(16)
daten:   DATA 0,1,2,3,124,125,126,127
         DATA 128,129,130,131,252,253,254,255
RESTORE daten
FOR i=1 TO 16: READ z(i): NEXT i: RETURN

holprog: anfad=SADD(code$)
p$="48E7E08041FA001010181218D00140D84CDF01074E75"
FOR i=0 TO 10
  POKEW anfad+2*i,VAL("&H"+MID$(p$,4*i+1,4))
NEXT i: RETURN
```

P.23.1 Die Flags des CCR (Amiga)

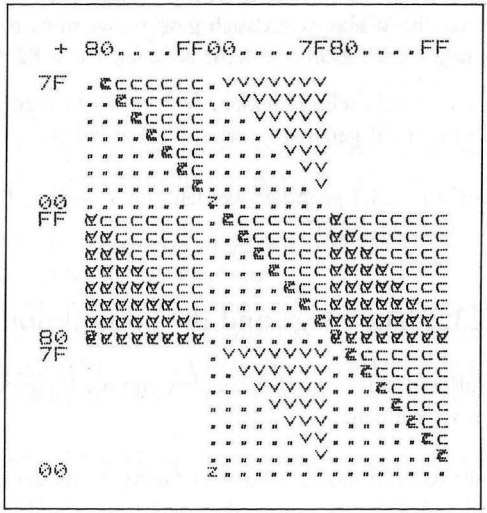


Bild 23.1 V- und C-Flag nach .B-Addition

Bild 23.1 zeigt dreimal mehr als bisher besprochen:

1. Bisher war nur von vorzeichenlosen Zahlen die Rede (Quadrat unten rechts im Bild). Die Seiten dieses Quadrats reichen von links 00 bis nach rechts FF und von unten 00 nach oben FF. Wie schon mehrfach angesprochen, lassen sich die Inhalte eines Bytes (usw.) aber auch als signed numbers auffassen (Quadrat oben links); die reichen dann von 80 (= −128) über FF, 00 usw. bis hin zu 7F (= 127). In einer hierauf beruhenden quadratischen Darstellung sähe die c-Verteilung etwas anders aus: Jeder Quadrant des einen Quadrats ist auch im anderen enthalten – nur in anderer Anordnung. Das veranlaßte zu der ornamentalen Verschmelzung der beiden möglichen Sichtweisen.
2. Das Z-Flag wird bei der Addition bekanntlich gesetzt, wenn das Ergebnis 0 ist: 7F + 81 = (1)00, 7E + 82 = (1)00 usw. Bild 23.1 zeigt auch das Z-Flag, jedoch mehr als Orientierungshilfe.
3. Das V-Flag (Überlaufbit, overflow) aus P.22.4 lieferte die Initialzündung für dieses Kapitel; es wird wohl nicht soviel benutzt wie die anderen, ist aber für die vorzeichenrichtige Rechnung (u.a.m.) unerläßlich. Es wird gesetzt, wenn a) zwei positive Zahlen eine im Sinne der Zweierkomple-

mentarithmetik negative Summe ergeben; Beispiel: 4A + 5B = A5: Das MSb ist gesetzt! b) V-Flag wird auch gesetzt, wenn zwei positive Summanden ein negatives Resultat liefern; Beispiel: 81 + 82 = (1)03.

Das X-Flag kann hier unbeachtet bleiben, denn es wird bei der Addition (und bei der Subtraktion auch) genauso gesetzt wie Carryflag.

A.23.1: Was muß an P.23.1 geändert werden, um a) V-Flag, b) Z-Flag, c) N-Flag und d) X-Flag anzuzeigen?

23.2 Die Flags und die Subtraktion

A.23.2: Was muß an P.23.1 geändert werden, um die Flags bei der Subtraktion beobachten zu können?

Für Bild 23.2 gilt weithin das zu Bild 23.1 Gesagte. Bei der Subtraktion ist jedoch die Reihenfolge der Operanden nicht gleichgültig, deshalb zur Legende: In 83 – F1 ist der Minuend 83 in der linken Randleiste zu suchen, der Subtrahend F1 in der oberen. Zu den einzelnen Flags:

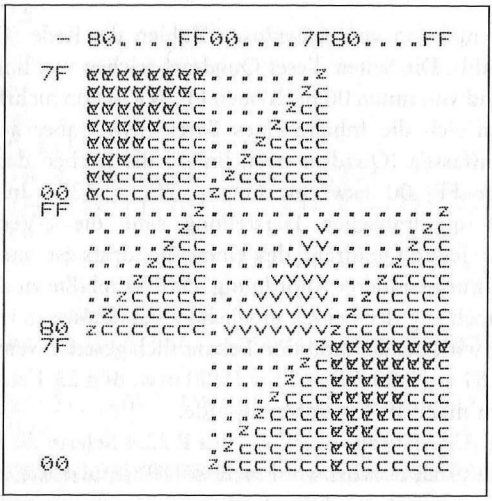


Bild 23.2 V- und C-Flag nach .B-Subtraktion

1. C-Flag: 83 − F1 = (−1)92. Hier findet ein (negativer) Übertrag (borrow, borgen) statt, und Carry wird gesetzt. Bei vorzeichenloser Betrachtung ist das leicht beschrieben: C-Flag wird stets dann gesetzt, wenn der Subtrahend größer als der Minuend ist. Bei Beachtung des Vorzeichens sind vier Fälle zu unterscheiden:
   a) «Positiv minus negativ»: Wie leicht einzusehen ist, wird Carry stets gesetzt: p1 − n2 = p1 −(100 − p2) p1 + p2 − 100, worin |n| ≤ $80 und |n| ≤ $7F.
   b) «Negativ minus positiv»: Carry wird nicht gesetzt, wohl aber (sozusagen statt dessen) das Überlaufbit.
   c) «Negativ minus negativ»: Hier wird Carryflag gesetzt, wenn der Subtrahend größer als der Minuend ist; Beispiel: AC − CA = (−1)E2 oder dezimal −84 − (−54) = −30, weil −84 < −54.
   d) «Positiv minus positiv»: Auch hier Übertrag, wenn Subtrahend zu groß.
2. Zeroflag wird bei Gleichheit der Operanden gesetzt; die Differenz ist dann 0.
3. V-Flag: Hier führt eine Fallunterscheidung letztlich auf die Definition zurück: Überlauf liegt vor, wenn das MSb des Zieloperanden von 0 nach 1 oder von 1 nach 0 wechselt.
4. Das N-Flag wird gesetzt wie das MSb des Zieloperanden bzw. des Minuenden; X-Flag wie C-Flag.

23.3 Die Flags und die Bedingungsschlüssel

Wenn man in P.23.1 die Addition durch CMP.B D0,D1 mit Opcode B200 ersetzt und das Ganze ein wenig verändert, erhält man P.23.2, dessen Ergebnisse wir vorwegnehmen (Bild 23.3).

Unter bereits begründeter Außerachtlassung des X-Flags verbleiben theoretisch 16 mögliche Flagkombinationen. Da aber eine Zahl nicht gleichzeitig negativ und gleich 0 sein kann, scheiden a priori vier aus. Bild 23.3 zeigt die verbleibenden 12 möglichen Bitmuster, daneben die ihnen entsprechenden Bedingungsschlüssel, wie sie in Befehlen wie Bcc, DBcc usw. eingesetzt werden, und die Anzahl ihres Vorkommens unter den 256 realisierten Vergleichen. Es fällt auf, daß weitere fünf Flagkombinationen nicht vorkommen. Die Erfahrung zeigt, daß z. B. V- und C-Flags gemeinsam nicht ohne ein

....	PL	NE	VC	CC	GE	GT	HI	56	
...C	PL	NE	VC	CS	GE	GT	LS	28	
..V.	PL	NE	VS	CC	HI	LE	LT	28	
..VC	PL	NE	VS	CS	LE	LS	LT	0	<<<
.Z..	PL	EQ	VC	CC	GE	LE	LS	16	
.Z.C	PL	EQ	VC	CS	GE	LE	LS	0	<<<
.ZV.	PL	EQ	VS	CC	LE	LS	LT	0	<<<
.ZVC	PL	EQ	VS	CS	LE	LS	LT	0	<<<
N...	MI	NE	VC	CC	HI	LE	LT	36	
N..C	MI	NE	VC	CS	LE	LS	LT	56	
N.V.	MI	NE	VS	CC	GE	GT	HI	0	<<<
N.VC	MI	NE	VS	CS	GE	GT	LS	36	
								256	

Bild 23.3 Die Flagkombinationen und die Bedingungsschlüssel

EQ				.Z..			
NE	....	...C	..V.		N...	N..C	N.VC
PL	....	...C	..V.	.Z..			
MI					N...	N..C	N.VC
VC	....	...C		.Z..	N...	N..C	
VS			..V.				N.VC
CC	....		..V.	.Z..	N...		
CS		...C				N..C	N.VC
GE	....	...C		.Z..			N.VC
GT	....	...C					N.VC
HI	....		..V.		N...		
LE			..V.	.Z..	N...		
LS		...C		.Z..		N..C	N.VC
LT			..V.		N...	N..C	

Bild 23.4 Die Bedingungsschlüssel und die Flagkombinationen

weiteres Flag, nämlich N, gesetzt werden. Das leuchtet ein, denn C wird nur gesetzt, wenn ein Übertrag stattfindet, V bei einem Wechsel des MSb des Zieloperanden. Hierfür einige Beispiele in Form der internen Subtraktion:

Ziel:	12	34	56	78	82	93	BA	FE
Quelle:	34	8F	65	FF	AF	BC	C3	FF
	DE	A5	F1	79	D3	D7	F7	FF

In allen 8 Beispielen tritt Carry auf, nur in den ersten 3 auch Überlauf, und das bei negativem Resultat. (Das ist natürlich kein mathematischer Beweis!) Vollständigkeitshalber wird auch die inverse Tabelle vorgelegt (Bild 23.4). Man erkennt aus ihr, mit welchen Flagkombinationen z.B. ein Bcc ausgeführt wird.

Wenn Sie nun noch einige konkrete Beispiele betrachten mögen, kann Ihnen dabei P.23.2 helfen.

```
DIM c$(11)
c$( 0)=".... PL NE VC CC GE GT HI"
c$( 1)="...C PL NE VC CS GE GT LS"
c$( 2)="..V. PL NE VS CC HI LE LT"
c$( 4)=".Z.. PL EQ VC CC GE LE LS"
c$( 8)="N... MI NE VC CC HI LE LT"
c$( 9)="N..C MI NE VC CS LE LS LT"
c$(11)="N.VC MI NE VS CS GE GT LS"
code$=STRING$(30,0): GOSUB holprog

job:
INPUT"0,Z ";h1$,h2$: POKE anfad+22,VAL("&H"+h1$)
                     POKE anfad+23,VAL("&H"+h2$)
  CALL anfad: pw=PEEK(anfad+25) AND 15
  PRINT"CMP.B "+h1$+","+h2$+" -> "c$(pw)
PRINT: GOTO job

holprog: anfad=SADD(code$)
p$="48E7E08041FA001010181218B20040D84CDF01074E75"
FOR i=0 TO 10
  POKEW anfad+2*i,VAL("&H"+MID$(p$,4*i+1,4))
NEXT i: RETURN
```

P.23.2 Flagkombinationen und Bedingungsschlüssel

24
Subtraktion und Fließkomma-Subtraktion

Das folgende Programm ist wieder zur Zusammenarbeit mit dem großen FP-Rahmenprogramm bestimmt. Um aber den zur Verfügung stehenden Raum sparsam zu nutzen, wird auch hier nur ein Torso entwickelt. Zunächst folgt ein Überblick über die Subtraktionsbefehle.

24.1 Die Subtraktionsbefehle

SUB.X <ea>,Dn und SUB.X Dn,<ea>
Diese Befehle subtrahieren den Quelloperanden vom Zieloperanden. Alle Flags werden regulär (so wie bisher dargestellt) betroffen.

SUBA.X <ea>,An
Ist das Ziel ein Adreßregister, wird SUB zu SUBA. Zugelassen sind nur .W und .L-Format. Flags werden von diesem Befehl nicht verändert.

SUBI.W #Konstante,<ea>
Je nach Formatangabe kann eine 8-, 16- oder 32-bit-Konstante vorgegeben werden. Die Flags werden regulär gesetzt.

SUBQ.X #Konstante,<ea>
Dieser Befehl ist schneller als die anderen, weil die 3-bit-Konstante bereits im Opcode enthalten ist; da eine Subtraktion von 0 nicht sinnvoll ist, codiert das Bitmuster 000 eine 8. Die Flags werden regulär beeinflußt.

SUBX.X Dq,Dz und SUBX.X −(Aq),−(Az)
Hier wird nicht nur der Quelloperand, sondern auch noch der Inhalt des eXtend-Bit vom Ziel subtrahiert. Man erhält also bei gleichen Operanden zwei verschiedene Ergebnisse, das größere, wenn X-Flag gelöscht ist, das kleinere, wenn bei der vorhergehenden Subtraktion geborgt wurde. Der Befehl ist also zur Aneinanderreihung bei Subtraktionen mit erhöhter Genauigkeit bestens geeignet.

Die Flags werden – bis auf Z – regulär beeinflußt. Z wird gelöscht, wenn eine Teildifferenz von 0 verschieden ist. Will man eine Gesamtdifferenz 0 erkennen, muß man also vorher Z-Flag setzen. Ist es nach der letzten Teilsubtraktion immer noch gesetzt, wurde es nie gelöscht; es trat also nie eine von 0 verschiedene Teildifferenz auf: Die Gesamtdifferenz ist also auch 0.

24.2 Problemanalyse

Da unsere Mantissen als Langwörter in einem (Daten-)Register Platz haben, werden unsere Subtraktionen in einer einzigen Operation abgeschlossen. Daher ist auch kein Übertrag aus einer vorhergehenden Operation zu berücksichtigen, und der geeignete Befehl ist SUB.L Dq,Dz.

Wie schon bei der FP-Addition erklärt, dürfen nur FP-Zahlen mit gleichen Exponenten voneinander subtrahiert werden. Die kleinere der beiden vorzeichenlosen Zahlen muß so lange nach rechts geshiftet werden, bis ihr Exponent dem der größeren gleich wurde. Heikler sind Überlauf und Übertrag. Hierfür zeigt Bild 24.1 zunächst einige konkrete Beispiele.

In der UPN, der wir hier folgen, ist \$41 – \$02 so zu schreiben: \$41 \$02 –. Der Minuend \$41 soll als erster eingegeben werden und rutscht bei Eingabe des Subtrahenden \$02 auf den zweiten Platz im Zahlenstapel, während die \$02 den ersten (TOS) einnimmt. Die linke Randleiste – auf ein Byte verkürzt – verzeichnet so die Mantisse des Operanden auf Platz zwei, den Minuend, während die obere einige mögliche Subtrahenden im TOS darstellt. Unsere Mantissen sind so normiert, daß – wir hatten uns auf positive Zahlen

–	01	02	3E	3F	40	41	7E	7F
7F	7E	7D	41	40	3F	3E	01	00
7E	7D	7C	40	3F	3E	3D	00	FF
41	40	3F	03	02	01	00	C3	C2
40	3F	3E	02	01	00	FF	C2	C1
3F					FF	FE	C1	C0
3E					FE	FD	C0	BF
02					C2	C1	84	83
01					C1	C0	83	82

Bild 24.1 Einige Beispiele für FP-Subtraktion

beschränkt – ihr MSb gelöscht, das nächstfolgende aber gesetzt ist. Durch Shiften zum Angleichen der Exponenten können bei Einleitung der Subtraktion mehrere der führenden Bits gelöscht sein. Hieraus erklärt sich die Auswahl der Kurzmantissen in Bild 24.1.

Die Subtraktion kann unter den gemachten Voraussetzungen dreierlei Resultate erbringen:

1. Das Resultat ist 0. (Dann sollte wohl auch der Exponent auf Null gesetzt werden.)
2. Das Resultat ist positiv: keine Probleme.
3. Das Resultat ist negativ. In diesem Falle sind, wie der Augenschein lehrt, N und V und C gesetzt. Wir wollen hier so verfahren, daß durch Negation der Absolutwert der Mantisse erzeugt wird; anschließend wird das MSb als Kennzeichen der Negativität gesetzt. In das Konvertierungsprogramm wäre eine entsprechende Erweiterung aufzunehmen.

Als Beispiel wird 18.2300 − 24.5630 = −6.3330 in FP-Notierung berechnet:

18.2300	0005 48EBE000
− 24.5630	− 0005 6240A000
= − 6.3330	= 0005E6AB4000
Durch Negation wird daraus	0005 1954C000
und nach Normieren	0003 65530000
und Setzen des Minusflags	0003 E5530000

Die Rückumwandlung des FP-Ergebnisses wird noch einmal ausführlich in Bild 24.2 vorgeführt.

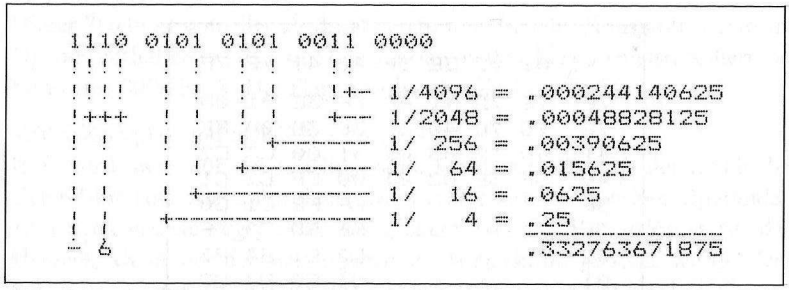


Bild 24.2 Rückverwandlung der negativen Differenz

24.3 Der Programmtorso

```
020000  48E7 F080        SUBTRA: MOVEM.L D0-D3/A0,-(A7)
020004  41FA 0060                LEA     ZALSTA(PC),A0
020008  3018                     MOVE.W  (A0)+,D0
02000A  2218                     MOVE.L  (A0)+,D1
02000C  3418                     MOVE.W  (A0)+,D2
02000E  2618                     MOVE.L  (A0)+,D3
020010  B440            VERGEX: CMP.W   D0,D2
020012  670E                     BEQ.S   SBTRKT
020014  6506                     BCS.S   D3RESH
020016  E281           D1RESH: ASR.L   #1,D1
020018  5240                     ADDQ.W  #1,D0
02001A  60F4                     BRA.S   VERGEX
02001C  E283           D3RESH: ASR.L   #1,D3
02001E  5242                     ADDQ.W  #1,D2
020020  60EE                     BRA.S   VERGEX
020022
020022  6100 0036        SBTRKT: BSR     DROP
020026  9681                     SUB.L   D1,D3
020028  670E                     BEQ.S   GLEICH
02002A  6410                     BCC.S   NIXNEG
02002C  4483                     NEG.L   D3
02002E  611C                     BSR.S   NORMIR
020030  0083 80000000            ORI.L   #$80000000,D3
020036  6006                     BRA.S   FERTIG
020038  7000           GLEICH: MOVEQ   #0,D0
02003A  6002                     BRA.S   FERTIG
02003C  610E           NIXNEG: BSR.S   NORMIR
02003E  41FA 001C      FERTIG: LEA     RESULT(PC),A0
020042  30C2                     MOVE.W  D2,(A0)+
020044  20C3                     MOVE.L  D3,(A0)+
020046  4CDF 010F                MOVEM.L (A7)+,D0-D3/A0
02004A  4E75                     RTS
```

P.24.1 Die FP-Subtraktion (I)

Zur Konzentration auf das Wesentliche wieder nur ein Programmrumpf. Er unterscheidet sich in folgendem von der FP-Addition:

1. ADD.L wurde zu SUB.L. Entsprechend wurden auch die Labels umbenannt.
2. Während bei der Addition die Reihenfolge der Summanden beliebig ist, muß es hier D1,D3 heißen. Dementsprechend müssen bei FERTIG D2 und D3 abgelegt werden. Eine spätere Zusammenfassung sollte auch die Zuordnung bei der Addition angleichen, damit möglichst viele Programmteile gemeinsam genutzt werden können.
3. Nach der Subtraktion ist zunächst auf Null-Ergebnis zu prüfen. Ggf. wird auch der Exponent auf Null gesetzt.
4. Wurde Carry gesetzt, muß die entstandene negative Differenz wie in der Analyse besprochen aufbereitet werden.
5. Während nach der Addition ein evtl. Überlauf auszugleichen war, ist hier auf das sozusagen Umgekehrte zu achten: Es ist möglich, daß die Mantisse mit mehreren Nullen beginnt. Dann muß durch Linksshiften normiert werden.

```
02004C  0803 001E       NORMIR: BTST    #30,D3
020050  6606                   BNE.S   NORMEX
020052  E383                   ASL.L   #1,D3
020054  5342                   SUBQ.W  #1,D2
020056  60F4                   BRA.S   NORMIR

020058  4E75           NORMEX: RTS
02005A
02005A  4E75           DROP:   RTS
02005C
02005C                 RESULT: DS.W    3
020062  3E3E3E3E       KENNER: DC.L    '>>>>'
020066  0005           ZALSTA: DC.W    $0005
020068  6240A000               DC.L    $6240A000
02006C  0005                   DC.W    $0005
02006E  48EBE000               DC.L    $48EBE000
```

P.24.1 Die FP-Subtraktion (II)

25
Vom K-Seka-Assembler (Amiga)

Der M68000 ist ein relativ neuer Prozessor, dementsprechend sind auch die M68000-Computer relativ neu, ebenso Tools, Software und Bookware – soweit überhaupt schon vorhanden. Insofern kann auch der Aufbau eines Buches «geschichtlich» bedingt sein.

25.1 Inbetriebnahme des K-Seka-Assemblers

Nachdem man seinen Amiga mit der Kickstart-Diskette bis zur ersten Stufe hochgefahren hat, verlangt er nach mehr: Auf dem Bildschirm wird grafisch die Workbench-Diskette angefordert. Hier ist die erste Klippe zu nehmen.

Selbstverständlich bewahrt man seine kostbaren Originaldisketten an einem sicheren Ort auf und verwendet Arbeitskopien. Der K-Seka-Assembler wird über das Command Line Interface (CLI) bedient. Offenbar befindet sich das entsprechende File nicht auf der Arbeitskopie der Workbench-Diskette, wenn man nicht vor dem Kopieren im Programmteil Preferences das Symbol CLI ON aktivierte. Ihre Arbeitskopie wird nun wahrscheinlich «Copy of Workbench» heißen. Der Assembler verlangt aber nach einer Diskette namens «Workbench».

Sie sollten nun das Symbol der Copy-of-Workbench-Diskette mit der linken Maustaste aktivieren (anklicken), dann mit der rechten Taste oben links am Bildschirmrand das Fensterchen «Workbench» öffnen, die Maus bis zur Option Rename nach unten bewegen und die rechte Taste loslassen. Jetzt können Sie die Diskette «Copy of Workbench» in «Workbench» umbenennen.

Klicken Sie nun das Symbol der neubenannten Diskette (rechts oben) zweimal an, und öffnen Sie im jetzt neu geöffneten Fenster durch Doppelklick die Schublade «System». Ein neues Fenster öffnet sich: «System». In ihm klicken Sie das Symbol «CLI» zweimal an. Ein weiteres Fenster wird

aufgetan: «New Cli Window», in ihm das Bereitschaftszeichen «1>». Tippen Sie nun «seka» (natürlich ohne die Anführungszeichen) und cr ein. CLI protestiert: «unknown command»; kein Wunder, denn woher soll der Amiga wissen, was das Kommando «seka» bedeutet?

Wir legen die Arbeitskopie der Seka-Diskette ein und rufen sie mit «seka» auf. Wieder wird gefensterlt: Zunächst öffnet sich ein großes Seka-Fenster und darin oben links ein kleineres mit der Aufforderung, doch bitte schön noch einmal die Workbench-Diskette einzulegen; offenbar werden von ihr weitere Programmteile für das im RAM stehende Betriebssystem benötigt. Wir tun auch das.

Nun erscheint im großen Seka-Fenster eine vierzeilige Vorstellung des Softwarepakets mit Copyright-Vermerk, darunter die Frage WORKSPACE KB>. Der Seka-Anleitung ist zu entnehmen, daß dem Assemblerpaket im Idealfall (wenn keine anderen Jobs laufen) 150 KB zugewiesen werden sollten. Wir tun's und sehen nun das Promptzeichen SEKA>.

25.2 Vom Seka-Editor

Zur Bearbeitung ein und desselben Textes stehen zwei Editoren zur Verfügung, der Screen-Editor und der Line-Editor. Der letztere eignet sich wohl mehr zur Erstellung längerer Programme; seine zusätzlichen Befehle erlauben es, den Cursor beispielsweise um n Zeilen nach oben oder unten zu bewegen, n Zeilen auf Bildschirm oder Drucker auszugeben u.a.m.

Wir wählen anfangs den Screen-Editor, der es erlaubt, mit den Cursorbewegungstasten beliebige Bildschirmpositionen anzusteuern und dort den Quelltext zu editieren (aufzustellen). Man gelangt in den Screen-Editor durch Drücken der Taste ESCape, und ebenfalls durch ESC wieder zurück ins große Seka-Fenster (Bild 25.1).

Der Quelltext soll erst im folgenden Kapitel inhaltlich besprochen werden; hier zum Formalen: Kommentare werden durch Semikolon (;) eingeleitet. Editor (und Assembler) nehmen auch Kleinbuchstaben an; das gilt sogar für die sechs oberen Hexadezimalziffern: (a, b ... f) = (A, B ... F).

Der Quelltext braucht nicht tabellarisch angeordnet zu sein, obgleich die Tabulatortaste dabei helfen könnte; zwischen Mnemonik und Operanden muß lediglich ein Platz frei bleiben (Space). Labels werden mit Doppelpunkt (:) abgeschlossen; nach diesem ist kein Space nötig.

```
 1 ;P.25.1 multiplizieren
 2
 3 movem.l d0-d1/a0-a1,-(a7)
 4 lea zahlen(pc),a0
 5 lea produk(pc),a1
 6 bsr.s multiu
 7 bsr.s multiu
 8 bsr.s multiu
 9 lea zahlen(pc),a0
10 bsr.s multis
11 bsr.s multis
12 bsr.s multis
13 movem.l (a7)+,d0-d1/a0-a1
14 rts
15
16 multiu:move.w (a0)+,d0
17 move.w (a0)+,d1
18 mulu d0,d1
19 bra.s ableg
20
21 multis:move.w (a0)+,d0
22 move.w (a0)+,d1
23 muls d0,d1
24 ableg:move.l d0,(a1)+
25 move.l d1,(a1)+
26 rts
27
28 zahlen:dc.w $1234
29 dc.w $5678
30 dc.w $2345
31 dc.w $FFFE
32 dc.w $FFFD
33 dc.w $FFFC
34 produk:blk.l 12
35
36 end
```

Bild 25.1 Der Quelltext (Seka)

Die meisten der zahlreichen Assemblerdirektiven entsprechen allgemeinem Brauch. Statt DS (define storage) reserviert blk Speicherraum und überschreibt ihn mit einem evtl. vorgegebenen Datum (vgl. P.25.2) oder dem Default-Wert FFFF. Auch beim Seka-Assembler darf das «end» nicht vergessen werden.

25.3 Das Assemblieren

Das Assemblieren ist denkbar einfach; noch einfacher geht's nicht! Wir verlassen den Editor mit ESC und drücken a. Übergehen wir nun die Eingabeaufforderung OPTIONS>, so wird verdeckt assembliert, und mit einigem Glück erhalten wir die Meldung «No Errors». Aber auch ein Fehler im Quelltext erfreut dank der komfortablen Korrekturhilfe: Nach ESC zeigt der Cursor sofort auf die beanstandete Stelle. Option v (view) gibt das Assemblerprotokoll auf dem Bildschirm aus, Option p auf dem Drucker. Natürlich kann man Quelltext und Objektcode auf Diskette aufzeichnen. Das aber wird nur den Amiga- und Seka-User interessieren; und der kann das der Seka-Anleitung entnehmen.

25.4 Der Programmablauf

In der vorliegenden Form ist das Assemblerprotokoll nur ein Codierdokument, das nicht zum direkten Programmablauf verhilft. Wir begeben uns nun zurück ins AmigaDOS, indem wir «!» drücken und dann die Frage Exit to System, Sure ? mit y (yes) beantworten. Nun ist wieder das CLI-Fenster inmitten der Workbench-Grafik zu sehen. Es wird durch Anklicken außerhalb und Anklicken des «Fenster-in-den-Hintergrund-Symbols» entfernt, ebenso die Workbench-Symbole.

Es sei dem Leser überlassen, P.25.1 über ein Hostprogramm oder per Monitor zum Ablauf zu bringen. NDR-User können das ja noch einfacher tun, und QL-User fügen vorher noch den Opcode 7000 ein. Auf jeden Fall aber sollte das Resultat zur späteren Auswertung im nächsten Kapitel, in dem auch die Multiplikation besprochen wird, festgehalten werden (Bild 25.2).

Eine andere Ablaufmöglichkeit sei gleich vorgestellt. P.25.2 unterscheidet sich wenig, und doch sehr wesentlich von P.25.1:

```
 1                      ;P.25.1 multiplizieren        -766-
 2
 3 000000 48E7C0C0              movem.l d0-d1/a0-a1,-(a7)
 4 000004 41FA0030              lea     zahlen(pc),a0
 5 000008 43FA0038              lea     produk(pc),a1
 6 00000C 6114                  bsr.s   multiu
 7 00000E 6112                  bsr.s   multiu
 8 000010 6110                  bsr.s   multiu
 9 000012 41FA0022              lea     zahlen(pc),a0
10 000016 6112                  bsr.s   multis
11 000018 6110                  bsr.s   multis
12 00001A 610E                  bsr.s   multis
13 00001C 4CDF0303              movem.l (a7)+,d0-d1/a0-a1
14 000020 4E75                  rts
15
16 000022 3018          multiu: move.w  (a0)+,d0
17 000024 3218                  move.w  (a0)+,d1
18 000026 C2C0                  mulu    d0,d1
19 000028 6006                  bra.s   ableg
20
21 00002A 3018          multis: move.w  (a0)+,d0
22 00002C 3218                  move.w  (a0)+,d1
23 00002E C3C0                  muls    d0,d1
24 000030 22C0          ableg:  move.l  d0,(a1)+
25 000032 22C1                  move.l  d1,(a1)+
26 000034 4E75                  rts
27
28 000036 1234          zahlen: dc.w    $1234
29 000038 5678                  dc.w    $5678
30 00003A 2345                  dc.w    $2345
31 00003C FFFE                  dc.w    $FFFE
32 00003E FFFD                  dc.w    $FFFD
33 000040 FFFC                  dc.w    $FFFC
34 000042 FFFFFFFF      produk: blk.l   12
35
36 000072                       end
```

P.25.1 Das erste Seka-Programm

```
04DC42  04DC 1234 0626 0060 04DC 2345 2344 B976
04DC52  04DC FFFD FFF9 000C 04DC 1234 0626 0060
04DC62  04DC 2345 FFFF B976 04DC FFFD 0000 000C
```

Bild 25.2 Der Speicherauszug zu Programm 25.1

Diesmal wird das Programm von Seka beim Betriebssystem angemeldet und muß dazu formale und inhaltliche Voraussetzungen erfüllen:

1. Einsprungadresse und Aussprungstelle sind durch Labels zu kennzeichnen. Der Einsprung wird i. a. mit START bezeichnet. Beim Ausgang hätte «end:» stehen können; so schlägt es Seka vor, und die beiden ends stören sich nicht: Das Label end würde mit seinem (relativen) Zahlenwert in die Symboltabelle aufgenommen werden; es steht ja auch in der Kolonne der Labels. Die Assemblerdirektive end steht in anderer Position im Quelltext und wird dort vom Assembler als ihn betreffender Befehl erkannt. Die Aussprungmarke muß nicht «end» oder «END» heißen; wir haben es mit Erfolg mit einem anderen Namen versucht.
2. Das Programm endet nicht mit dem üblichen RTS, sondern mit dem Aufruf einer Betriebssystemroutine, die die Rückkehr bewerkstelligt. Es kann nicht dafür garantiert werden, daß das Programm so auf allen Amigas läuft; es ist denkbar, daß die Aufrufadresse in anderen Betriebssystemversionen an anderer Stelle liegt. Dann kann man es so versuchen: MOVE.L 4,A6 JSR −144(A6), womit man den Rücksprung über einen Vektor mit Adreßdistanz finden kann. Wenn auch das nicht hilft, lese man in der Seka-Anleitung nach. Hier wird die vereinfachte Darstellung vorgezogen.
3. Die aus AmigaDOS übernommenen Inhalte der benutzten Register brauchen offenbar nicht auf den Stapel gerettet zu werden; für die Sicherung dieser Daten sorgt entweder Seka oder AmigaDOS.
4. D1 muß vor der Rückkehr nach AmigaDOS auf Null gesetzt werden.

Nach erfolgreichem Assemblieren kann man das Programm so starten:

```
SEKA>g start                (goto start)
BREAKPT>suedpol             (und dann bis Südpol)
BREAKPT>      (cr)          (keine weiteren Breakpoints)
```

```
 1                          ;P.25.2 multiplizieren        -769-
 2
 3 000000 41FA0032      start:   lea      zahlen(pc),a0
 4 000004 43FA003A               lea      produk(pc),a1
 5 000008 6116                   bsr.s    multiu
 6 00000A 6114                   bsr.s    multiu
 7 00000C 6112                   bsr.s    multiu
 8 00000E 41FA0024               lea      zahlen(pc),a0
 9 000012 6114                   bsr.s    multis
10 000014 6112                   bsr.s    multis
11 000016 6110                   bsr.s    multis
12 000018 7200                   moveq    #0,d1
13 00001A 4EB900000490 suedpol:  jsr      $0490
14
15 000020 3018         multiu:   move.w   (a0)+,d0
16 000022 3218                   move.w   (a0)+,d1
17 000024 C2C0                   mulu     d0,d1
18 000026 6006                   bra.s    ableg
19
20 000028 3018         multis:   move.w   (a0)+,d0
21 00002A 3218                   move.w   (a0)+,d1
22 00002C C3C0                   muls     d0,d1
23 00002E 22C0         ableg:    move.l   d0,(a1)+
24 000030 22C1                   move.l   d1,(a1)+
25 000032 4E75                   rts
26
27 000034 1234         zahlen:   dc.w     $1234
28 000036 5678                   dc.w     $5678
29 000038 2345                   dc.w     $2345
30 00003A FFFE                   dc.w     $FFFE
31 00003C FFFD                   dc.w     $FFFD
32 00003E FFFC                   dc.w     $FFFC
33 000040 43686970     produk:   blk.l    16,$43686970
34
35 000080                        end
```

P.25.2 Die unmittelbar lauffähige Variante von P.25.1

Nach dem Programmablauf werden die Registerinhalte (wie schon beim NDR-Computer besprochen) angezeigt.

Um nun die Ergebnisse zu kontrollieren, fordert man mit q.w start (query = befragen) einen Speicherauszug ab Label start an. Die Ergebnisse sind die gleichen wie bei P.25.1, doch stehen sie an anderer Stelle.

26
Multiplikation und Fließkomma-Multiplikation

Nach dem Seka-Exkurs stellen wir eine weitere Routine für ein FP-System bereit.

26.1 Die Multiplikationsbefehle

```
                          unsigned          signed
1234 * 5678   =   06260060   =   06260060
2345 * FFFE   =   2344B976       FFFFB976
FFFD * FFFC   =   FFF9000C       0000000C
```

Bild 26.1 Die Resultate von Programm 25.1 (bzw. 26.2)

Der M68000 stellt zwei Multiplikationsbefehle zur Verfügung:

MULU <ea>,Dn (Unsigned Multiply) multipliziert zwei .W-Daten als vorzeichenlose Zahlen,

MULS <ea>,Dn (Signed Multiply) mit Vorzeichen.

Für den Quelloperanden sind alle regulären Adressierungsarten – bis auf An – gestattet.

Das .L-Produkt füllt das ganze Ziel-Datenregister. Wie aus Bild 25.2 zu ersehen ist, bleibt der Quelloperand erhalten; in unserem speziellen Fall bleibt das MSW des Quell-Datenregisters unverändert. (Es hat verständlicherweise in P.25.1 und P.25.2 aus der unterschiedlichen Umgebung heraus auch unterschiedliche Inhalte.)

A.26.1:
a) Woher stammen die Teil«ergebnisse» (?) $04DC in Bild 25.2?
b) Wie läßt sich die Vermutung experimentell bestätigen?

Das erste Beispiel ist schnell gedeutet: Das Produkt zweier positiver Zahlen ist signed oder unsigned positiv und ist in beiden Fällen gleich.

Zum zweiten Beispiel: Unter Berücksichtigung des Vorzeichens ist $2345 * (−2) = −$468A, und $100000000 − $468A = $FFFFB976.

Das dritte Beispiel ist einfach zu verstehen: Bei Berücksichtigung der Vorzeichen ergibt sich (−3) * (−4) = +12.

Die Multiplikationsbefehle lassen X unberührt, setzen V und C auf Null und N und Z dem Resultat entsprechend.

26.2 Mehrstellige Multiplikation

Nun ist das Problem zu lösen, daß unsere Mantissen Langwörter sind, während MULU und MULS nur W.-Faktoren aufnehmen können. Wir erinnern uns unserer Schulkenntnisse und wandeln die «schriftliche Multiplikation» ein bißchen ab (Bild 26.2).

Damit wäre des Guten zuviel getan, denn von den errechneten 16 Stellen passen nur 8 in die Mantisse hinein. Es bleibt zu fragen, ob man das LSW*LSW-Teilprodukt fortlassen darf. Dagegen sprechen zwei Gründe: Da wegen unserer Norm das Produkt um eine oder zwei Binärstellen nach links geshiftet werden muß, würde auf mehr als nur den evtl. Übertrag zur niederwertigsten Stelle verzichtet werden. Zudem wäre die Arbeitseinsparung nur gering; beiläufig soll ja auch eine vollständige 32*32=64-bit-Multiplikation vorgeführt werden.

```
                                    1234ABCD * 567890EF
                                    -------------------
      ABCD *      90EF =                       6143B463
  1234     *      90EF =                   A4E3E8C
      ABCD * 5678      =                   3A076618
  1234     * 5678      =               6260060
                                    -------------------
                       =               062644B605E7B463
```

Bild 26.2 Die «schriftliche Multiplikation»

Man kann das Problem nun so lösen, daß nicht mehr Register als bisher benötigt werden. Die Operanden bleiben ja bis zum DROP im Zahlenstapel verfügbar; man könnte von da den jeweils benötigten Bestandteil mittels Adressierungsart «Adreßregister indirekt mit Adreßdistanz» holen.

A.26.2: Schreiben Sie ein solches Programm!

Wir gehen den anderen Weg, der leicht auf weniger spezielle Vorgaben übertragen werden kann. Dabei wird versucht, das problemtypische Swappen übersichtlich auszuführen.

Zur Registerdisposition (Bild 26.3) ist zu vermerken, daß nur ein Exponent nach D0 geholt zu werden braucht. Der andere kann sofort aus dem Zahlenstapel heraus hinzuaddiert werden. D2 steht also für die Multiplikation der Mantissen zur Verfügung. Hierzu werden auch noch D4 und D5 herangezogen; das stört später evtl. übergeordnete Programmteile nicht, wenn sie nur während der Subroutine MULTIP auf den A7-Stapel gerettet werden.

In Bild 26.3 symbolisieren Großbuchstaben Datenwörter, AB z.B. das Langwort mit dem MSW A und dem LSW B. Die erste Multiplikation verknüpft das LSW(D5) = B mit dem LSW(D2) = D und legt das .L-Produkt

```
                D1   D2   D3   D4   D5
Holen:          AB        CD
Kopieren:            CD             CD   AB
MULU D5,D2           bd
SWAP D1         BA
SWAP D3                   DC
MULU D3,D1      ac
MULU D5,D3                bc
ADDSWA               bc
SWAP D4                             DC
MOVE D4,D3                DC
SWAP D3                   CD
SWAP D5                                  BA
MULU D5,D3                ad
ADDSWA               ad
```

Bild 26.3 Registerdisposition bei der mehrstelligen Multiplikation

bd in D2 ab. Daß fünf Zeilen später das Produkt bc zwischen zwei Register geschrieben wurde, soll symbolisieren, daß das MSW des Produkts bc von der Subroutine ADDSWA zu D1, das LSW zu D2 addiert wird. Mit dieser Legende dürfte der Algorithmus hinreichend erklärt sein.

26.3 Die Fließkomma-Multiplikation (Amiga)

Da MULTIP fast unverändert auf andere Rechner übertragen werden kann, soll diesmal der Amiga benutzt werden. Als Beispiel ist vorgesehen

```
12,75 * 6,5 = (0004 66000000) * (0003 68000000)
              (0007 66000000*68000000)
               0007 29700000
               0006 52E00000
Verf'Korr.:    0007 52E00000
```

Unsere spezielle Nutzung von Bit31 als Vorzeichenträger bedingt eine abschließende Verfahrenskorrektur.

P.26.1 folgt penibel dem in Bild 26.3 dargestellten Rechengang, so daß abschließend nur noch einmal auf den AmigaDOS-spezifischen Schluß verwiesen wird. A6 wird auf dem schon unter Abschnitt 25.4 (2.) angesprochenen Weg mit einer Basisadresse geladen, von der aus die tatsächliche Einsprungadresse zur Rückkehr-Routine über eine Adreßdistanz bestimmt wird. Dieser Offset Exit wurde mit einer EQU-Direktive vorgegeben, wofür Seka auch das Gleichheitszeichen akzeptiert. Dieser Assembler erwartet aber statt MOVEA ein schlichtes MOVE.

```
 1                          ;
 2
 3                          ;P.26.1 FP-Multiplikation
 4
 5          FFFFFF70        Exit    =       -144
 6
 7 000000 41FA005C          multip: lea     zalsta(pc),a0
 8 000004 3018                      move.w  (a0)+,d0
 9 000006 2218                      move.l  (a0)+,d1
10 000008 D058                      add.w   (a0)+,d0
11 00000A 2618                      move.l  (a0)+,d3
12 00000C 2403                      move.l  d3,d2
13 00000E 2803                      move.l  d3,d4
14 000010 2A01                      move.l  d1,d5
15 000012 C4C5                      mulu    d5,d2
16 000014 4841                      swap    d1
17 000016 4843                      swap    d3
18 000018 C2C3                      mulu    d3,d1
19 00001A C6C5                      mulu    d5,d3
20 00001C 611A                      bsr.s   addswa
21 00001E 4844                      swap    d4
22 000020 2604                      move.l  d4,d3
23 000022 4843                      swap    d3
24 000024 4845                      swap    d5
25 000026 C6C5                      mulu    d5,d3
26 000028 610E                      bsr.s   addswa
27 00002A 0801001E         normul: btst    #30,d1
28 00002E 6616                      bne.s   ablage
29 000030 D482                      add.l   d2,d2
30 000032 D381                      addx.l  d1,d1
31 000034 5340                      subq.w  #1,d0
32 000036 60F2                      bra.s   normul
33
34 000038 4842             addswa:  swap    d2
35 00003A D443                      add.w   d3,d2
36 00003C 4842                      swap    d2
37 00003E 4843                      swap    d3
38 000040 48C3                      ext.l   d3
```

```
39 000042 D383                    addx.l   d3,d1
40 000044 4E75                    rts
50 000046 5240          ablage:   addq.w   #1,d0
51 000048 61000012                bsr      drop
52 00004C 30C0                    move.w   d0,(a0)+
53 00004E 20C1                    move.l   d1,(a0)+
54
55 000050 7200          hinaus:   moveq    #0,d1
56 000052 2C7900000004            move.l   4,a6
57 000058 4EAEFF70      multex:   jsr      Exit(a6)
58
59 00005C 4E75          drop:     rts
60
61 00005E 0004          zalsta:   dc.w     $0004
62 000060 66000000                dc.l     $66000000
63 000064 0003                    dc.w     $0003
64 000066 68000000                dc.l     $68000000
65 00006A 0000                    dc.w     $0000
66 00006C 00000000                dc.l     $00000000
67
68                        ;P.26.1 FP-Multiplikation (II)
69
70 000070                         end
```

P.26.1 FP-Multiplikation für Amiga

27
Division und Fließkomma-Division

Mit dieser vierten Grundrechenart soll die Serie der FP-Routinen beendet werden. Interessenten werden sicherlich in der Lage sein, den Ansatz weiter fortzuführen.

27.1 Die beiden Divisionsbefehle

DIVS <ea>,Dn und DIVU <ea>,Dn bewirken eine vorzeichenrichtige bzw. -lose Division. Wie bei den MULx-Befehlen sind für den Quelloperanden alle Adressierungsarten außer An zugelassen. X-Flag wird von der Division nicht verändert, C immer auf Null gesetzt, N und Z dem Resultat entsprechend. Das V-Flag zeigt ggf. einen Überlauf an:

DIVS (bzw. DIVU) teilt den .L-Zieloperanden durch den .W-Quelloperanden. Der ganzzahlige Quotient steht zum Schluß im LSW des Ziel-Datenregisters, der Divisionsrest im MSW. Versucht man nun, einen sehr großen Dividenden durch einen sehr kleinen Divisor zu teilen, tritt Überlauf ein, wenn der Quotient .W-Format überschreitet. In diesem Falle wird das V-Flag gesetzt, und die Operanden werden nicht verändert.

Komplizierter wird's bei der bekanntlich ausgeschlossenen Division durch null. In diesem Falle geht der Prozessor in einen sog. Ausnahmezustand (exception) über, der tief in das Betriebssystem hineinführt. Hierauf können wir erst nach der Besprechung der TRAP-Befehle eingehen. Man täte ohnehin gut daran, diesen Fall durch entsprechende Abfrage zu vermeiden.

27.2 Noch ein Primzahlprogramm

Um eine konkrete Anwendung eines Divisionsbefehls aufzuzeigen, folgt ein weiteres Primzahlprogramm: Übersichtlichkeitshalber lassen wir die beiden ersten Primzahlen, 2 und 3, und die geraden Zahlen «außen vor». So sind nur noch die Zahlen 5, 7, 9, ... auf Teilbarkeit durch 3, 5, 7, ... zu prüfen. In einer äußeren Schleife, die mit dem Laden des Teilerregisters D1 mit dem «Urteiler» 3 beginnt, wird die stets in D0 stehende auf Teilbarkeit zu prüfende Zahl um jeweils 2 erhöht.

Es schließt sich die innere Schleife OBWOHL (ob die Zahl wohl teilbar ist?) an. Sie quadriert den momentan erreichten Teiler im Rechenregister D2. Ist das Quadrat des Teilers größer als die zu prüfende Zahl geworden, wurde kein möglicher Teiler und damit eine Primzahl (in D0) gefunden; ihre Ausgabe bewirkt die Subroutine PRIMUT (utgäwen). Danach wird die zu prüfende Zahl um 2 erhöht und wieder der Ur-Teiler geladen.

Hat aber das Quadrat des Teilers den Prüfling noch nicht übertroffen, muß der Divisionsrest ermittelt werden. Ist er 0, so ist die geprüfte Zahl nicht prim und ist um 2 zu erhöhen; danach weiter bei URTEIL. Bei von 0 verschiedenem Divisionsrest muß mit dem nächsten Divisor (NXTDIV) eine neue Prüfung durchgeführt werden.

Die Division liefert schließlich auch die Abbruchbedingung. Sobald der Prüfling 3 * 65536 = 196608 = $30000 überschreitet, führt die Division $30000 : 3 zum Überlauf. V-Flag = 1 führt zurück ins Grundprogramm.

PRIMUT: Da D0 für die Ausgabe benötigt wird, muß sein Inhalt auf den Stapel (oder anderweitig) gerettet werden. Die Aufbereitung der Hexadezimalzahl zum Dezimalstring übernimmt in bereits bekannter Weise @PRINT8D, die ziffernweise Ausgabe @CO2. Die Ausgabe verdient eine nähere Besprechung. Ungebremst wäre sie so schnell, daß es unmöglich wäre, sie zu verfolgen. Eine Warteschleife hilft nur wenig. Außerdem würde die Bildschirmkapazität nur zu einem Zehntel ausgenutzt werden. Wir formatieren daher die Ausgabe auf 8 Schreibpositionen, so daß in jeder Zeile rechtsbündig zehn Primzahlen erscheinen. Das Bild ist jetzt angenehm; mehrere Varianten sind möglich.

Die Ausgabe wird so formatiert: CMPI.B #0,(A1)+ sucht den Puffer nach der von @PRINT8D gelieferten Schlußnull ab. Wurde sie gefunden, zeigt A1 auf das auf den Schlußcode folgende Byte. Der Ausgabepointer A0 wird auf A1 – 9 gesetzt, wo eine ausreichende Anzahl blanks ($20) bereitgestellt wurde. Jetzt wird jede Ausgabe acht Zeichen lang.

```
020000
020000  4EB8 3FEC       PRIMZL: JSR     @CLRSCREEN.S
020004  7005                    MOVEQ   #5,D0
020006  7203              URTEIL: MOVEQ   #3,D1
020008  3401              OBWOHL: MOVE.W  D1,D2
02000A  C4C2                    MULU    D2,D2
02000C  B480                    CMP.L   D0,D2
02000E  6306                    BLS.S   TEILEN
020010  6116                    BSR.S   PRIMUT
020012  5480              NXTZAL: ADDQ.L  #2,D0
020014  60F0                    BRA.S   URTEIL
020016
020016  2400              TEILEN: MOVE.L  D0,D2
020018  84C1                    DIVU    D1,D2
02001A  690A                    BVS.S   EXIT
02001C  4842                    SWAP    D2
02001E  4A42                    TST.W   D2
020020  67F0                    BEQ.S   NXTZAL
020022  5441              NXTDIV: ADDQ.W  #2,D1
020024  60E2                    BRA.S   OBWOHL
020026  4E75              EXIT:   RTS
020028
020028  2F00              PRIMUT: MOVE.L  D0,-(A7)
02002A  41FA 0030                 LEA     BUFFER(PC),A0
02002E  4EB8 15BE                 JSR     @PRINT8D.S
020032  43FA 0028                 LEA     BUFFER(PC),A1
020036  0C19 0000         SUCH:   CMPI.B  #0,(A1)+
02003A  66FA                      BNE.S   SUCH
02003C  41E9 FFF7                 LEA     -9(A1),A0
020040  1018             HOLCHR: MOVE.B  (A0)+,D0
020042  6708                    BEQ.S   WARTEN
020044  4EB8 0AE0              JSR     @CO2.S
020048  60F6                    BRA.S   HOLCHR
02004A  78FF                    MOVEQ   #-1,D4
02004C  51CC FFFE         WARTEN: DBRA    D4,WARTEN
020050  201F                    MOVE.L  (A7)+,D0
020052  4E75                    RTS
020054
020054  20202020                DC.L    $20202020
020058  20202020                DC.L    $20202020
02005C                    BUFFER: DS.B    12
020068
020068                            END
```

P.27.1 Noch ein Primzahlprogramm

27.3 Bitweise Division

Die Befehle DIVU und DIVS erlauben eine Langwort-durch-Wort-Division. Auf das Beispiel von Bild 26.2 bezogen, würde das bedeuten $62644B60 : $1234 = $567BC statt $56789.0EF. Der Fehler ist beträchtlich; außerdem tritt Überlauf ein. Eine .L:.L-Division ist nur umständlich zu realisieren. Mit einer bitweisen Division (wie schon in Abschnitt 22.5 für einen Spezialfall umrissen) läßt sich Abhilfe schaffen.

Bei der FP-Division liegen die Dinge noch etwas einfacher, weil die Operanden schon normiert vorliegen; beide beginnen nach der führenden Null mit einem gesetzten Bit. Wir kehren nun das Zahlenbeispiel aus Abschnitt 26.3 um:

$$\begin{aligned} 82{,}875 : 12{,}75 &= (0007\ 52E00000) : (0004\ 66000000) \\ &= (0003\ 52E00000{:}66000000) \end{aligned}$$

Bild 27.1 verkürzt nun der Übersicht halber die .L- zur .W-Division; in Wirklichkeit wird stets mit 32 bit gerechnet. Der Quotient wird Bit für Bit gewonnen: Ist der Divisor größer als der momentane Dividend, wird an den Quotienten eine Null angefügt und der Dividend um eine Stelle nach links geshiftet. Ist er nun größer oder gleich dem Divisor geworden, wird eine Eins angehängt, subtrahiert und geshiftet, wie aus Bild 27.1 zu ersehen. Die

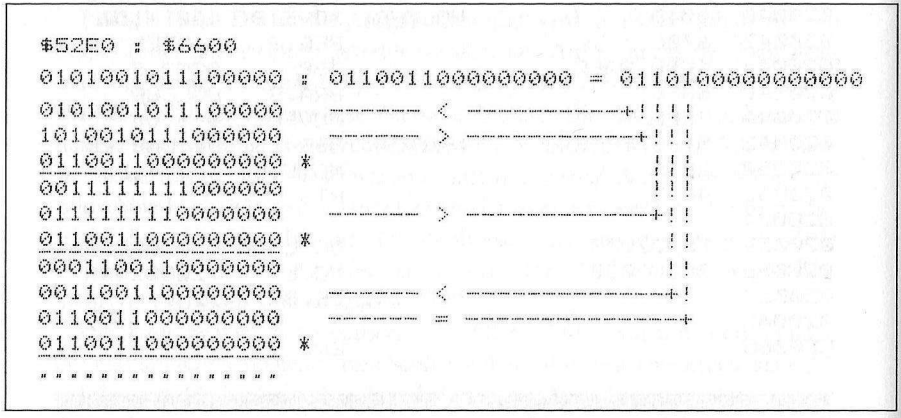


Bild 27.1 Die bitweise Division

Sternchen kennzeichnen darin den (hier zu subtrahierenden) Divisor. Die Vergleichsoperatoren sind so zu lesen:

geshifteter Dividend <,=,> Divisor

P.27.2 zeigt die auf ihre künftige Verwendung hin vorangepaßte Routine. Der Quotient wird Bit für Bit in D0 aufgebaut, wobei folgende Schwierigkeit auftritt: Ganz gleich, ob die 0- oder 1-Bits von rechts durch ROXL oder ADDX eingeschoben werden, Zugang gibt es nur über das X-Flag, das der Befehl CMP.L D1,D3 nicht beeinflußt, und SUB.L D1,D3 im falschen Sinn. Ausweg: Nach der Subtraktion wird X-Flag mit EORI invertiert. Dann muß

```
020000  41FA 002A              LEA     ZALSTA(PC),A0
020004  2218                   MOVE.L  (A0)+,D1
020006  671E                   BEQ.S   NULDIV
020008  2618                   MOVE.L  (A0)+,D3
02000A  671C                   BEQ.S   NULQUO
02000C  7000                   MOVEQ   #0,D0
02000E  44FC 0010      VERGL:  MOVE    #$10,CCR
020012  B681                   CMP.L   D1,D3
020014  6B02                   BMI.S   NOTABZ
020016  9681                   SUB.L   D1,D3
020018  0A3C 0010      NOTABZ: EORI    #$10,CCR
02001C  D180                   ADDX.L  D0,D0
02001E  D683                   ADD.L   D3,D3
020020  4A83                   TST.L   D3
020022  66EA                   BNE.S   VERGL
020024  20C0           FERTIG: MOVE.L  D0,(A0)+
020026  4E71           NULDIV: NOP
020028  4E71           NULQUO: NOP
02002A  4E75                   RTS
02002C
02002C  66000000       ZALSTA: DC.L    $66000000
020030  52E00000               DC.L    $52E00000
020034  00000000               DC.L    $00000000
020038                         END
```

P.27.2 Die bitweise Division

es also, wenn eine Null eingeschoben werden soll, vor der Umkehrung gesetzt gewesen sein. Das bewirkt das MOVE to CCR zu Beginn des VERGLeichs. Die Routine läuft zunächst zufriedenstellend; die vorgegebene Division liefert ganz richtig 0000000D, was natürlich noch zu 68000000 hochzushiften ist. Wie aber ist dabei der Exponent zu verändern? Wir gehen dieser Frage nach und verringern den Dividenden um 1 bit; $52E00000 wird zu $52DFFFFF. Das Programm verweilt in einer endlosen Schleife.

A.27.1:
a) weshalb?
b) Schaffen Sie Abhilfe!

Mit der Abhilfe erhält man folgende Quotientenserie:

52E00000 : 66000000 = 68000000 (vorher 0000000D)
52DFFFFF : 66000000 = 67FFFFFE
52DFFFF0 : 66000000 = 67FFFFEB
52DFFF00 : 66000000 = 67FFEBE
52DFF000 : 66000000 = 67FFEBEB
52DF0000 : 66000000 = 67FEBEBE

Offenbar ist gar nicht die Position des Quotienten-Bitmusters im Langwort für die Interpretation des Ergebnisses ausschlaggebend, sondern nur die Ziffernfolge. Die hätte man aber durch eine einheitliche Abfrage gleich in die richtige Position bringen können. Damit wird P.27.2 zu P.27.3. Die Sternchen weisen auf die Veränderungen hin:

Wie aber verhält sich nun das geänderte Programm gegenüber Veränderungen des Divisors? Dazu sei daran erinnert, daß nicht jede beliebige Division zugelassen ist, sondern immer unsere Norm Bit31 = 0 und Bit30 = 1 vorausgesetzt wird. P.27.3 liefert folgende Ergebnisse:

52E00000 : 65F00000 = 681052DF
52E00000 : 65FF0000 = 68010507
52E00000 : 65FFF000 = 68001050
52E00000 : 65FFFF00 = 68000105
52E00000 : 65FFFFF0 = 68000010
52E00000 : 65FFFFFF = 68000001
52E00000 : 66000000 = 68000000

```
020000  41FA 0030              LEA     ZALSTA(PC),A0
020004  2218                   MOVE.L  (A0)+,D1
020006  6724                   BEQ.S   NULDIV
020008  2618                   MOVE.L  (A0)+,D3
02000A  6722                   BEQ.S   NULQUO
02000C  7000                   MOVEQ   #0,D0
02000E  44FC 0010      VERGL:  MOVE    #%10,CCR
020012  B681                   CMP.L   D1,D3
020014  6B02                   BMI.S   NOTABZ
020016  9681                   SUB.L   D1,D3
020018  0A3C 0010      NOTABZ: EORI    #%10,CCR
02001C  D180                   ADDX.L  D0,D0
02001E  0800 001E              BTST    #30,D0   *
020022  6606                   BNE.S   FERTIG   *
020024  D683                   ADD.L   D3,D3
020026  4A83                   TST.L   D3
020028  60E4                   BRA.S   VERGL    *
02002A  20C0           FERTIG: MOVE.L  D0,(A0)+
02002C  4E71           NULDIV: NOP
02002E  4E71           NULQUO: NOP
020030  4E75                   RTS
020032
020032  66000000       ZALSTA: DC.L    $66000000
020036  52E00000               DC.L    $52E00000
02003A  00000000               DC.L    $00000000
02003E                         END
```

P.27.3 Bitweise Division mit einheitlichem Abbruch

Nun ist nur noch ein kleines Problem zu lösen. Unter vorläufig weiterer Außerachtlassung der Exponenten berechnet P.27.3 folgende vier Eck-Quotienten:

a) 40000000 : 7FFFFFFF = 40000000 sollte sein 1/2
b) 40000000 : 40000000 = 40000000 sollte sein 1
c) 7FFFFFFF : 7FFFFFFF = 40000000 sollte sein 1
d) 7FFFFFFF : 40000000 = 7FFFFFFF sollte sein 2

Während die Quotienten b) ... d) untereinander im richtigen Verhältnis stehen, fällt a) aus der Reihe, da die neue Abbruchbedingung in jedem Falle zu einem gesetzten Bit30 führt.

A.27.2:

a) Wie könnte diese Ungleichförmigkeit korrigiert werden? Abfrage zur Fallunterscheidung?

b) Bauen Sie die korrigierte und vervollständigte Routine in P.22 ein.

27.4 Die Vervollständigung des FP-Rechenwerks

Es sei nun dem interessierten Leser überlassen, die anderen drei Grundrechenarten in P.22 einzubinden. Damit kann man z.B. nach HERON (ca. 100 v. Chr.) Quadratwurzeln durch Intervallschachtelung ziehen und damit wiederum nach EUKLID (ca. 300 v. Chr.) die Kreiszahl Pi berechnen und damit Sinuswerte und damit ... Letztlich fehlt nur noch eine Abfrage auf Gleichheit. Viel Freude bei der Realisierung!

28
Die Türme von Hanoi

Die Erzählung sei so wiedergegeben, wie sie der Verfasser gehört zu haben glaubt:

In einem Tempel in Hanoi sind seit längerem Priester damit beschäftigt, einen kegelähnlichen Stapel («Turm») von 60 zylindrischen Scheiben unterschiedlicher Größe zu versetzen. Dies soll nach folgender Vorschrift geschehen: «Man lege immer nur eine Scheibe um, und zwar so, daß niemals eine größere auf einer kleineren abgelegt wird. Danach ist bis zur nächsten vollen Stunde zu beten». Der Tempelboden bietet gerade dreimal soviel Platz, wie die größte Scheibe benötigt. Wenn der Turm so von seinem ursprünglichen Platz zum gegenüberliegenden und dort zu seiner ursprünglichen Form umgeschichtet wurde, ist das Ende der Welt gekommen.

Das Problem dient als Paradebeispiel einer rekursiven Lösung, die sich sehr komfortabel in einer höheren Programmiersprache formulieren läßt. Für die Wahl der Maschinensprache gibt es u.a. zwei Gründe: Sie kommt noch dichter an das eigentliche «Wie» heran. Für eine Darstellung in hochauflösender Grafik könnte die eine oder andere höhere Sprache zu langsam sein.

28.1 Problemanalyse

Angenommen (Bild 28.1), es sollten drei Scheiben von 1 nach 3 versetzt werden, dann könnte man so vorgehen: Man muß ja schließlich einmal die Scheibe 3, die unterste, von 1 nach 3 transportieren. Da man sie aber nur einzeln bewegen darf, setzt man zuerst das Teiltürmchen 1–2 auf die freie Position 2. Dann darf man die Scheibe 3 aus der Anfangsposition 1 zur Endposition 3 verrücken. Zum Schluß ist dann «nur noch» das Teiltürmchen 1–2 draufzusetzen. Und damit man das Rezept nicht vergißt, schreibt man es auf: 212 113 223. Diese Triaden sind so zu lesen:

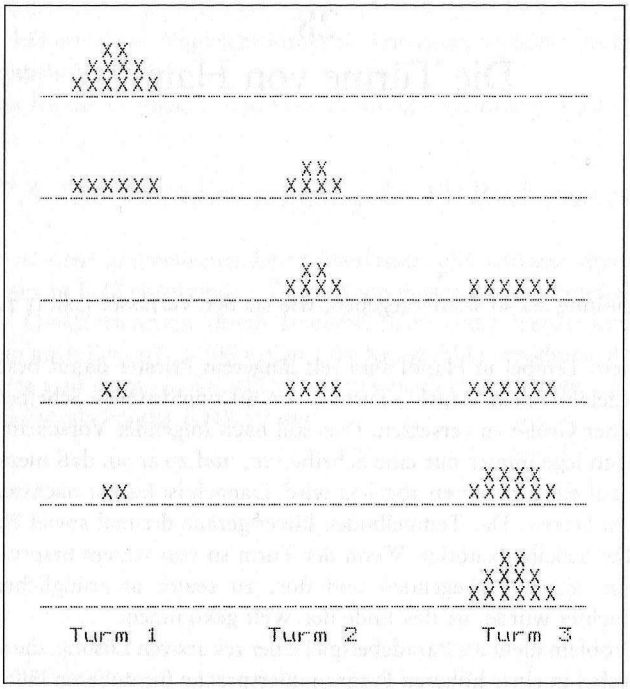


Bild 28.1 Türme von Hanoi – Strategie

Bring 2 Scheiben von Position 1 nach Position 2;
bring 1 Scheibe von Position 1 nach Position 3;
bring 2 Scheiben von Position 2 nach Position 3.

Der aufmerksame Leser wird ungeduldig seinen Einwand einbringen wollen: Teiltürmchen (hier aus zwei Scheiben bestehend) dürfen doch gar nicht umgesetzt werden! – Wir tun's ja auch noch nicht, sondern tun ja nur so, als ob wir's täten, und lösen nun das reduzierte Problem: Wie darf man ein aus zwei Scheiben bestehendes Türmchen von 1 nach 2 umsetzen? Da dabei die oberste Scheibe zum Freimachen der unteren (Nummer 2) zwischengelagert werden muß, kommt dafür nur der unbetroffene Platz 3 in Frage. Damit ergibt sich folgende Sequenz: 113 112 132. Das Ganze läßt sich auch so darstellen:

313
212 113 223
113 112 132

Nun beginnt unser momentan fünfstelliges Konzept mit vier ausführbaren Anweisungen: Setze je eine Scheibe von 1 nach 3, von 1 nach 2, von 3 nach 2, von 1 nach 3 um. Nun ist nur noch ein Teilproblem zu lösen, das wir gleich so notieren:

223
121 123 113

Damit haben wir die Aufgabe in sieben Zügen erledigt. Nota bene: $7 = 3 * (3 - 1) + 1$.

A.28.1:

a) Schreiben Sie die Sequenz für das Umlegen von vier Scheiben von Turm 2 nach Turm 1.
b) Wie viele «Züge» sind dazu erforderlich?
c) Stimmt die Notabene-Formel?

Da eine entsprechende Zugfortschreibung im Speicher besser zum freien Ende, also zu den höheren Adressen hin erfolgt, sollten wir das Erreichte rechtzeitig gespiegelt zu lesen üben. Bild 28.2 zeigt Aufbau und Abarbeitung der Befehlssequenz im Speicher. Man muß (z.B. die erste Zeile) von rechts her lesen: Transportiere drei Scheiben von Turm 1 nach Turm 3.

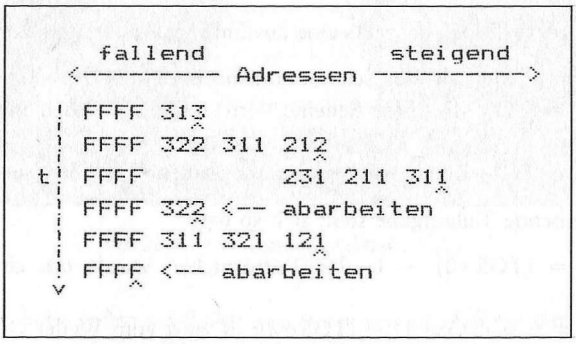


Bild 28.2 Die Befehlssequenz («∧» = TOS (Top Of Sequence))

Im folgenden Programm wird der Speicherbereich, in dem so die Arbeitsanweisungen erzeugt werden, REIHE genannt. Die Befehlssequenz beginnt oder endet – je nach Lesart – mit dem Schlußzeichen FFFF. Als REIHE-Pointer verwenden wir A4. Erreicht A4 bei TRIAL ein FF des Schlußzeichens, ist die Sequenz abgearbeitet, ist der Turm – hoffentlich – umgesetzt.

Damit ist bereits ein wesentlicher Teil der Strategie erkannt: Ist (TOS) = 1, dann bedeutet das «Leg eine Scheibe ...»; das ist erlaubt, also gleichbedeutend mit dem Befehl ZIEHeN. (Die freiwillige Verkürzung der Labels spart bei Benutzung der TAB-Taste bis zu drei Schreibpositionen ein.)

Ist hingegen (TOS) > 1, so heißt das ReKURS. Da A4 (als sozusagen globaler Pointer) stets auf TOS zeigt, läßt sich diese Fortschreibung wie folgt darstellen:

	3	1	3						← vor Umformung
TOS	−2	−1	+0	+1	+2	+3	+4	+5	+6	
	3	2	2	3	1	1	2	1	2	← nach Umformung

Diese Rekursion ist das Herzstück des Programms und soll daher genau besprochen werden.

Für das jüngste Tripel gilt:

(TOS+6) = (TOS+0) − 1; da wir bei jedem Denkschritt eine Scheibe liegen lassen, verringert sich die Höhe des Teiltürmchens jeweils um 1.
(TOS+5) = (TOS−1); das Teiltürmchen kommt von derselben Quelle.
(TOS+4) = 6 − (TOS−1) − (TOS−2); da die Summe der drei Turmnummern 6 beträgt, findet man so am bequemsten den Weder-Quelle-noch-Ziel-Platz.

Für das mittlere Tripel, das stets eine ausführbare Anweisung darstellt, gilt:

(TOS+3) = 1, weil wir *eine* Scheibe transportieren dürfen.
(TOS+2) = (TOS−1); diese Scheibe wird aus der gleichen momentanen Quelle geholt.
(TOS+1) = (TOS−2); sie wird zum momentanen (Teil-)Ziel verschoben.

Die verbleibende Teilaufgabe stellt sich so dar:

(TOS+0) = (TOS+0) − 1; das Teiltürmchen wurde um eine Scheibe niedriger.
(TOS−1) = 6 − (TOS−1) − (TOS−2); es wird vom Weder-Quelle-noch-Ziel-Platz zum
(TOS−2) = (TOS−2) unveränderten Ziel bewegt.

Sie finden diese Rekursionsvorschrift nahezu wörtlich bei RKURS in die Assemblersprache übersetzt; die Offsets sind dabei unmittelbar die Distanzen bei der Adressierung d(A4). Abschließend muß A4 zum neuen TOS zeigen, also um 6 erhöht werden.

28.2 Zur grafischen Darstellung auf dem Bildschirm

Das angestrebte Programm soll das Versetzen der Scheiben als feingrafische Bewegung auf den Bildschirm des QL (und später auch des NDR-Computers) bringen.

Für den QL bedeutet eine Lösung als Maschinenprogramm rund viermal so viele Bildpunkte, schnelleren Ablauf und dadurch kontinuierlichere Bewegungen als bei einem SuperBASIC-Programm (Bild 28.3).

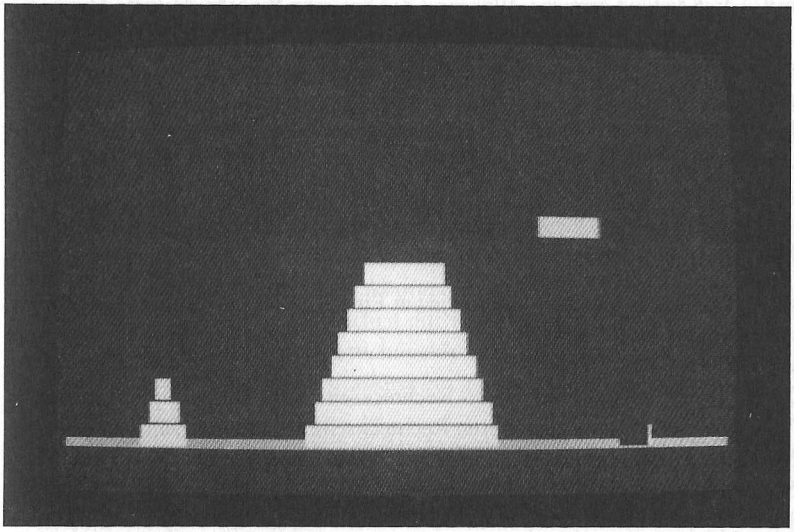


Bild 28.3 Die Türme von Hanoi (QL)

Beide Programme sollen möglichst kongruent sein, so daß nur eins für beide zu besprechen ist. Hieraus können sich im Detail für den einen oder anderen etwas umständlichere Lösungen oder nicht ganz plausible Koordinatenaus-

wahlen ergeben. Beim NDR-Computer entsprechen die Bildschirmkoordinaten der schulüblichen Cartesischen Ebene; beim QL zählt aber y vom oberen Bildschirmrand nach unten. Zur Vereinheitlichung wurde die Inversion dem NDR-Computer übertragen.

Die Scheiben werden als Rechtecke dargestellt. Dafür gibt es beim QL den Screen-Driver-TRAP SD_FILL; beim NDR-Computer wird das Rechteck mit der ebenfalls sehr schnellen Turtlegrafik aufgebaut. Einzelheiten finden Sie in den betreffenden Programmabschnitten.

28.3 Das Variablen-Handling

Unser Programm benötigt weit mehr Variable, als Register vorhanden sind. Die Variablen sind als .W-Daten im Arbeitsbereich des Programms ab EBER untergebracht und werden mittels Adressierungsart Adreßregister indirekt mit Adreßdistanz erreicht. Während der Sequenz-Pointer A4 (vgl. Abschnitt 28.1) mit der Sequenz in Sechserschritten wächst und in Dreierschritten schrumpft, ist der Variablenpointer A3 fest auf die SCHeibenDeFinition gerichtet. Die für jede Aufgabe veränderlichen Angaben ANZAhL der Scheiben, VONTUrm BISTUrm und VERZögerung werden mit negativen Distanzen angesprochen; so werden sie beim Löschen des Arbeitsbereichs zu Anfang eines jeden Programmdurchlaufs nicht überschrieben.

Diese Adreßdistanzen werden eingangs durch EQU-Statements den Variablennamen zugewiesen. Dann folgen:

die BREITe der Scheiben, die für jeden Einzeltransfer (in Zehnerstufen) neu zu berechnen ist;
die DICKE (stets 9 Pixel);
XVOLI, der Abstand der linken Rechteckseite vom linken Bildschirmrand;
YVOOB, der Abstand der oberen Rechteckseite vom oberen Bildschirmrand;
SCHNR, die Nummer der zu bewegenden Scheibe (stets ist BREIT = 10 * SCHNR);
MAXHO, die Höhe des momentan höchsten Turms (die Scheiben müssen ja hoch genug fliegen, aber auch nicht unnötig hoch);
HEBHO gibt an, um wie viele Scheibendicken das Transportgut zu heben ist, um MAXHO zu erreichen;

ZURSE sagt an, wie weit die Scheibe ZUR SEite verschoben werden muß (ein negatives ZURSE bedeutet «nach rechts»);
NACH Unten bildet sowohl den Abschluß der A3-Variablen als auch des Bewegungsablaufs.

28.4 Die vorbereitenden Routinen

Das Hauptprogramm ist nur 28 Bytes lang: Bildschirm und Pointer werden INITialisiert, die die Aufgabe bestimmenden Parameter EINgeLeSen, der Urturm intern codiert. Bei TRIAL wird – wie in Abschnitt 28.1 dargelegt – TOS abgefragt; steht dort FF, geht es zurück ins Betriebssystem oder (s. P.28.2) zurück nach BASIC; wir werden dort sehen, weshalb das Programm wieder – wenn auch recht elegant – in einen BASIC-Rumpf eingebettet wird. Ist aber die Aufgabe noch nicht abgearbeitet, muß das jüngste Anweisungstripel ANALYsiert werden.

INIT identifiziert zunächst einmal das dem Programm zugeordnete Fenster, setzt die beiden Pointer A3 und A4, löscht den Arbeitsbereich, zeichnet den Tempelboden und setzt das Sequenz-Schlußzeichen an den Anfang.

Zu SCREeN: $10001 ist die Kanalkennzahl des BASIC-Ausgabefensters; dieses wird im folgenden gelöscht: Mit (dem immer noch später zu besprechenden) TRAP #3 werden die Screen-Driver-Routinen erreicht, mit Vorwahl $20 in D0 das CLS zuzuordnende Pendant der Maschinenebene; die korrekte Bezeichnung SD_CLEAR mußte aus Platzgründen zu QCLR verkürzt werden. In D3 wurde dieser Aktion eine Zeit von 10 Fünfzigstelsekunden zugestanden (was ja wohl reichen dürfte); kann der Bildschirm in dieser Zeit aus irgendwelchen Gründen nicht gelöscht werden, wird die Ausführung eingestellt, so daß z.B. im Multitasking ein anderer Job Rechenzeit zugeteilt bekommen kann.

Zu LOEScH: Nach dem Löschen des Arbeitsbereichs muß die Angabe der (konstanten) Scheibendicke wieder eingespeichert werden.

KASTN zeichnet das Rechteck, was höchst elegant SD_FILL erledigt, wenn folgende Parameter richtig vorgegeben wurden: Kanalkennzahl – wie besprochen – in A0; Farbe in D1 (INIT gab hierfür 4, d.h. einen Grünton vor); timeout in D3 wie eben; Pointer zum Definitionsblock ist A1 (ebenfalls in INIT eingestellt). Die vier Parameter bei BODEN haben folgende Bedeutung: 1. Wort: Breite des Bodens = 436 Pixel. 2. Wort: Dicke des Fußbodens

```
                                * TvHanoi_t

        =0020                   QCLR   EQU      $20
        =002E                   QFILL  EQU      $2E

        =FFF8                   ANZAL  EQU      -8
        =FFFA                   VONTU  EQU      -6
        =FFFC                   BISTU  EQU      -4
        =FFFE                   VERZ   EQU      -2
                                * A3 = SCHDF
        =0000                   BREIT  EQU      0
        =0002                   DICKE  EQU      2
        =0004                   XVOLI  EQU      4
        =0006                   YVOOB  EQU      6
        =0008                   SCHNR  EQU      8
        =000A                   MAXHO  EQU      10
        =000C                   HEBHO  EQU      12
        =000E                   ZURSE  EQU      14
        =0010                   NACHU  EQU      16

0000'   6100 001A              START  BSR      INIT
0004'   6100 0066                     BSR      EINLS
0008'   6100 009A                     BSR      TURM0
000C'   0C14 FFFF              TRIAL  CMPI.B   #-1,(A4)
0010'   6706                        BEQ.S    BASIC
0012'   6100 00EA                     BSR      ANALY
0016'   60F4                        BRA.S    TRIAL
0018'   7000                    BASIC  MOVEQ    #0,D0
001A'   4E75                           RTS

001C'   6100 0020              INIT   BSR      SCREN
0020'   47FA 02E4                     LEA      SCHDF,A3
0024'   49FA 0322                     LEA      REIHE,A4
0028'   6100 0022                     BSR      LOESH
002C'   43FA 02C8                     LEA      BODEN,A1
0030'   7204                        MOVEQ    #4,D1
0032'   6100 002A                     BSR      KASTN
0036'   28FC FFFF FFFF              MOVE.L   #-1,(A4)+
003C'   4E75                           RTS
```

P.28.1 Die Türme von Hanoi (I)

```
003E'  207C 0001 0001  SCREN  MOVEA.L  #$10001,A0
0044'  760A                   MOVEQ     #10,D3
0046'  7020                   MOVEQ     #QCLR,D0
0048'  4E43                   TRAP      #3
004A'  4E75                   RTS

004C'  41D3                LOESH LEA        (A3),A0
004E'  701F                   MOVEQ     #31,D0
0050'  4298                LOPLO CLR.L      (A0)+
0052'  51C8 FFFC              DBRA      D0,LOPLO
0056'  377C 0009 0002         MOVE.W    #9,DICKE(A3)
005C'  4E75                   RTS

005E'  207C 0001 0001  KASTN  MOVEA.L  #$10001,A0
0064'  760A                   MOVEQ     #10,D3
0066'  702E                   MOVEQ     #QFILL,D0
0068'  4E43                   TRAP      #3
006A'  4E75                   RTS

006C'  196B FFF9 0002  EINLS  MOVE.B    ANZAL+1(A3),2(A4)
0072'  196B FFFB 0001         MOVE.B    VONTU+1(A3),1(A4)
0078'  18AB FFFD              MOVE.B    BISTU+1(A3),(A4)
007C'  49EC 0002              LEA       2(A4),A4
0080'  6100 0010              BSR       HOLTV
0084'  1014                   MOVE.B    (A4),D0
0086'  4880                   EXT.W     D0
0088'  14C0                   MOVE.B    D0,(A2)+
008A'  14C0                LEINT MOVE.B    D0,(A2)+
008C'  51C8 FFFC              DBRA      D0,LEINT
0090'  4E75                   RTS

0092'  302B FFFA           HOLTV MOVE.W    VONTU(A3),D0
0096'  6004                   BRA.S     HOLT2
0098'  302B FFFC           HOLTB MOVE.W    BISTU(A3),D0
009C'  E940                HOLT2 ASL.W     #4,D0
009E'  45F3 0002              LEA       2(A3,D0.W),A2
00A2'  4E75                   RTS
```

P.28.1 Die Türme von Hanoi (II)

(Höhe des Rechtecks) = 4 Pixel. 3. Wort: Abstand vom linken Bildrand = 6 Pixel. 4. Wort: Y-Abstand vom oberen Bildrand = 176 Pixel. Bitte beachten Sie, daß die vier Parameter der SCHeibenDeFinition in der gleichen Reihenfolge angeordnet sind, so daß durch bloßes Umsetzen von A1 auch die Scheibenrechtecke mit KASTN gezeichnet werden können.

EINLS: Nachdem vor dem Programmstart die die Aufgabe beschreibenden Daten in den nicht gelöschten Arbeitsbereich eingePOKEt wurden, werden sie nun in ihre weiteren Bestimmungsorte kopiert: Die Anzahl der umzulegenden Scheiben nach (dem künftigen) TOS, die Nummer des Quellturms nach TOS–1 und die des Zielturms nach TOS–2, woraufhin der Sequenz-Pointer A4 auf das neue TOS gerichtet wird.

Unser Programm kann in der vorgestellten Form bis zu 15 Scheiben im Quellturm erfassen. Im Arbeitsbereich sind dreimal 16 Bytes für die interne Codierung der momentanen Turmzustände reserviert. Wenn nun die Aufgabe lautet: «Versetze 5 Scheiben von Turm 3 nach Turm 1», dann wird ab TURM3 eingetragen 5 5 4 3 2 1 0 0 0 ... Die erste 5 zeigt an, daß der Quellturm momentan fünf Scheiben hoch ist, was z.B. für die Berechnung der Flughöhe wichtig ist. Diese fünf Scheiben liegen so: unten die Scheibe Nr. 5, darauf die Nr. 4 usw.

Vor dem Eintrag der Urturm-Codierung wird seine Adresse von der Routine HOLTV nach A2 geladen: D0 erhält die Nummer des Von-Turms, die versechzehnfacht mit Adreßdistanz 2 zu SCHDF addiert wird: So ergibt sich, wenn VON TUrm3 aus umgelegt werden soll, seine Adresse zu 2 + $0306 + $10 * 3 = $0338. (HOLTB lädt A2 analog mit der Adresse des Zielturms.) Damit dürften nun auch Absicht und Funktion der LEINT-Schleife deutlich geworden sein.

28.5 Angewandte Grafik und Analyse

Die Routine TURM0 zeichnet den Quellturm auf den Bildschirm. Die Adresse VONTU wird um 1 erhöht, womit über A2 die Nummer der ersten Scheibe nach D0 geholt werden kann. Falls einmal ein Spaßvogel null Scheiben versetzen möchte, wird die Routine ohne Ausführung verlassen. (Weitere Prüfungen auf Zulässigkeit der Abfragen sind nicht vorgesehen.)

Trägt aber die unterste Scheibe eine zulässige Nummer, dann werden aus ihr die erforderlichen Parameter für KASTN errechnet:

```
00A4'  6100 FFEC          TURM0  BSR       HOLTV
00A8'  45EA 0001                 LEA       1(A2),A2
00AC'  1012                      MOVE.B    (A2),D0
00AE'  6700 004C                 BEQ       EXTRM
00B2'  4880                      EXT.W     D0
00B4'  C0FC 000A                 MULU      #10,D0
00B8'  3740 0000                 MOVE.W    D0,BREIT(A3)
00BC'  322B FFFA                 MOVE.W    VONTU(A3),D1
00C0'  C2FC 0096                 MULU      #150,D1
00C4'  0441 004B                 SUBI.W    #75,D1
00C8'  E248                      LSR.W     #1,D0
00CA'  9240                      SUB.W     D0,D1
00CC'  3741 0004                 MOVE.W    D1,XVOLI(A3)
00D0'  377C 00AA 0006            MOVE.W    #170,YVOOB(A3)
00D6'  48A7 E000          LOPT0  MOVEM.W   D0-D2,-(A7)
00DA'  43D3                      LEA       (A3),A1
00DC'  7205                      MOVEQ     #5,D1
00DE'  6100 FF7E                 BSR       KASTN
00E2'  4C9F 0007                 MOVEM.W   (A7)+,D0-D2
00E6'  046B 000A 0000            SUBI.W    #10,BREIT(A3)
00EC'  670E                      BEQ.S     EXTRM
00EE'  066B 0005 0004            ADDI.W    #5,XVOLI(A3)
00F4'  046B 000A 0006            SUBI.W    #10,YVOOB(A3)
00FA'  60DA                      BRA.S     LOPT0
00FC'  4E75               EXTRM  RTS

00FE'  0C14 0001          ANALY  CMPI.B    #1,(A4)
0102'  673A                      BEQ.S     ZIEHN
0104'  1014               RKURS  MOVE.B    (A4),D0
0106'  5300                      SUBQ.B    #1,D0
0108'  1880                      MOVE.B    D0,(A4)
010A'  1940 0006                 MOVE.B    D0,6(A4)
010E'  196C FFFF 0002            MOVE.B    -1(A4),2(A4)
0114'  196C FFFF 0005            MOVE.B    -1(A4),5(A4)
011A'  196C FFFE 0001            MOVE.B    -2(A4),1(A4)
0120'  7006                      MOVEQ     #6,D0
0122'  902C FFFF                 SUB.B     -1(A4),D0
0126'  902C FFFE                 SUB.B     -2(A4),D0
012A'  1940 0004                 MOVE.B    D0,4(A4)
012E'  1940 FFFF                 MOVE.B    D0,-1(A4)
0132'  197C 0001 0003            MOVE.B    #1,3(A4)
0138'  49EC 0006                 LEA       6(A4),A4
013C'  4E75                      RTS
```

P.28.1 Die Türme von Hanoi (III)

1. Scheibenbreite = Nummer mal zehn.
2. Linke Kante der Scheibe: Rund 450 Pixel der Bildschirmbreite werden ausgenutzt; dann ist der Abstand zwischen zwei Turmmitten 150, der Abstand von Turm(mitte) 1 vom linken Rand 75. Also errechnet sich XVOLI aus 150 * Turm# − 75 − halbe Breite.
3. Die Scheibendicke wurde als konstant 9 vorgegeben.
4. Die unterste Scheibe ist 170 Pixel vom oberen Bildschirmrand entfernt.

Nun werden in der Schleife LOPT0 die einzelnen Scheiben gezeichnet und die Angaben für die jeweils nächste berechnet. Die beiden MOVEMs sind hier nicht erforderlich, wurden aber aus Kongruenzgründen (NDR) beibehalten. An KASTN müssen nun noch die Adresse der Scheibendefinition in A1 und die Farbe (Hellblau = 5) in D1 übergeben werden.

Fortzuschreiben sind folgende Parameter:

Die BREITe muß je weitere Scheibe um 10 verringert werden; wurde sie zu 0, ist der Turm fertig gezeichnet.
Der Seitenabstand XVOLI muß jeweils um 5, das ist die halbe Scheibenbreitenabnahme, vergrößert werden.
Der Abstand vom oberen Bildschirmrand ist um 10 zu verringern, so daß die einzelnen Scheiben durch eine ein Pixel breite Linie getrennt werden.

Zur ANALYse: Ist (EOS)=1, darf ein Zug ausgeführt werden, anderenfalls ReKURSion, wie schon in Abschnitt 28.1 ausführlich erklärt.

28.6 Der Scheibentransport

Ein Zug wird hier in vier Hauptabschnitten realisiert: Berechnung der SCHeibenDaTen und der KAStenPArameter, Ermittlung der Maximalhöhe und der Hebe-, Schiebe- und Absenkdistanz (HMAXD) sowie der eigentlichen Bewegung.

Damit die Kastenparameter berechnet werden können, muß SCHDT zunächst einmal die Kenn-Nummer des Quellturms, die des Zielturms und die Nummer der zu bewegenden Scheibe an die richtigen Adressen umspeichern. Die Abmessungen des «Kastens» werden fast wie in TURM0 berechnet. Es ist nicht die Aufgabe dieses Buches, perfekte Lösungen zum bloßen Abtippen vorzulegen, vielmehr sollen Erklärungen und Anregungen gegeben werden. In diesem Sinne möge der Leser prüfen, ob sich die Berechnung der Kastenparameter wirksam straffen läßt.

```
013E'  6100 0010      ZIEHN BSR      SCHDT
0142'  6100 0034            BSR      KASPA
0146'  6100 0068            BSR      HMAXD
014A'  6100 00B2            BSR      BEWEG
014E'  4E75                 RTS

0150'  102C FFFF      SCHDT MOVE.B   -1(A4),D0
0154'  4880                 EXT.W    D0
0156'  3740 FFFA            MOVE.W   D0,VONTU(A3)
015A'  102C FFFE            MOVE.B   -2(A4),D0
015E'  4880                 EXT.W    D0
0160'  3740 FFFC            MOVE.W   D0,BISTU(A3)
0164'  6100 FF2C            BSR      HOLTV
0168'  1012                 MOVE.B   (A2),D0
016A'  4880                 EXT.W    D0
016C'  1232 0000            MOVE.B   0(A2,D0.W),D1
0170'  4881                 EXT.W    D1
0172'  3741 0008            MOVE.W   D1,SCHNR(A3)
0176'  4E75                 RTS

0178'  302B 0008      KASPA MOVE.W   SCHNR(A3),D0
017C'  C0FC 000A            MULU     #10,D0
0180'  3740 0000            MOVE.W   D0,BREIT(A3)
0184'  322B FFFA            MOVE.W   VONTU(A3),D1
0188'  C2FC 0096            MULU     #150,D1
018C'  0441 004B            SUBI.W   #75,D1
0190'  E248                 LSR.W    #1,D0
0192'  9240                 SUB.W    D0,D1
0194'  3741 0004            MOVE.W   D1,XVOLI(A3)
0198'  6100 FEF8            BSR      HOLTV
019C'  1012                 MOVE.B   (A2),D0
019E'  4880                 EXT.W    D0
01A0'  C0FC 000A            MULU     #10,D0
01A4'  323C 00B4            MOVE.W   #180,D1
01A8'  9240                 SUB.W    D0,D1
01AA'  3741 0006            MOVE.W   D1,YVOOB(A3)
01AE'  4E75                 RTS
```

P.28.1 Die Türme von Hanoi (IV)

```
01B0'  102B 0012       HMAXD MOVE.B   18(A3),D0
01B4'  B02B 0022             CMP.B    34(A3),D0
01B8'  6404                  BCC.S    HMAX1
01BA'  102B 0022             MOVE.B   34(A3),D0
01BE'  B02B 0032       HMAX1 CMP.B    50(A3),D0
01C2'  6404                  BCC.S    HMAX2
01C4'  102B 0032             MOVE.B   50(A3),D0
01C8'  4880            HMAX2 EXT.W    D0
01CA'  3740 000A             MOVE.W   D0,MAXHO(A3)
01CE'  6100 FEC2             BSR      HOLTV
01D2'  302B 000A             MOVE.W   MAXHO(A3),D0
01D6'  9012                  SUB.B    (A2),D0
01D8'  5200                  ADDQ.B   #1,D0
01DA'  4880                  EXT.W    D0
01DC'  3740 000C             MOVE.W   D0,HEBHO(A3)
01E0'  6100 FEB6             BSR      HOLTB
01E4'  302B 000A             MOVE.W   MAXHO(A3),D0
01E8'  9012                  SUB.B    (A2),D0
01EA'  4880                  EXT.W    D0
01EC'  3740 0010             MOVE.W   D0,NACHU(A3)
01F0'  302B FFFA             MOVE.W   VONTU(A3),D0
01F4'  906B FFFC             SUB.W    BISTU(A3),D0
01F8'  3740 000E             MOVE.W   D0,ZURSE(A3)
01FC'  4E75                  RTS

01FE'  6100 0010       BEWEG BSR      HEBEN
0202'  6100 003A             BSR      SEITL
0206'  6100 009E             BSR      SENKE
020A'  6100 00C8             BSR      BUCHE
020E'  4E75                  RTS

0210'  3C2B 000C       HEBEN MOVE.W   HEBHO(A3),D6
0214'  CCFC 000A             MULU     #10,D6
0218'  43D3            LOPHE LEA      (A3),A1
021A'  7202                  MOVEQ    #2,D1
021C'  6100 FE40             BSR      KASTN
0220'  046B 0001 0006        SUBI.W   #1,YVOOB(A3)
0226'  43D3                  LEA      (A3),A1
0228'  7205                  MOVEQ    #5,D1
022A'  6100 FE32             BSR      KASTN
022E'  3A2B FFFE             MOVE.W   VERZ(A3),D5
0232'  4E71            WARTH NOP
0234'  51CD FFFC             DBRA     D5,WARTH
0238'  51CE FFDE             DBRA     D6,LOPHE
023C'  4E75                  RTS
```

P.28.1 Die Türme von Hanoi (V)

Die Maximalhöhe wird dadurch ermittelt, daß zuerst die Höhe von Turm 1 nach D0 geholt und mit h(T2) verglichen wird. Ist h(T2) ≤ h(T1), wird h(T2) ignoriert, anderenfalls übernommen – entsprechend h(T3). Falls ein Leser das Programm auf mehr als 15 Scheiben erweitern möchte, wird er die Adreßdistanzen 18, 34 und 50 beachten müssen.

Wie hoch muß nun die zu hebende Scheibe angehoben werden, damit Kollisionen vermieden werden? A2 wird auf den Quellturm (HOLTV) gerichtet und die in (A2) eingeschriebene Höhe von der Maximalhöhe subtrahiert; das ist HEBHO. Ganz entsprechend wird berechnet, um wie viele Scheibendicken NACH Unten abzulegen ist. Die ZUR-SEite-Bewegung ist schließlich durch die Differenz der beiden Turmnummern bestimmt.

Die BEWEGung wird unterteilt in Anheben, Seitenverschiebung, Absenken und Buchung der Veränderung.

Eine Scheibe wird optisch gehoben, indem sie in ihrer alten Position zunächst gelöscht und dann ein Pixel höher wieder hingezeichnet wird. HEBHO * 10 (Kastendicke plus Zwischenraum) gibt an, wie oft das geschehen muß (äußere Schleife). So würde aber der Kasten (man denke an die Geschwindigkeit eines Maschinenprogramms; es ist bis zu 300mal schneller als eine BASIC-Version) viel zu schnell bewegt werden und dabei nur schlecht zu sehen sein. Deshalb wird noch eine (innere) Warteschleife eingeschlossen. Das NOP kompensiert die gelegentlich von Assemblern zu Unrecht verweigerte Berechnung eines Rückwärtssprungs um −2. Damit dabei das Rechteck auch wirklich gehoben wird, muß jedesmal YVOOB um 1 vermindert werden.

Je nach Vorzeichen der Differenz Quellturm# – Zielturm# wird nach links oder rechts verschoben, was analog zum Anheben geschieht. Auch das Absenken wird nun nach kurzem Einblick verständlich sein.

Voraussetzung für den korrekten Fortgang ist die korrekte Buchung der Veränderung: Die Höhe des Quellturms ist um 1 zu verringern. Die Kennzahl der entfernten Scheibe braucht hingegen nicht gelöscht zu werden, denn sie liefert für das Weitere keine Information, sondern wird vielleicht einmal überschrieben. Ähnlich muß zur Höhe des Zielstapels 1 addiert werden; außerdem aber muß die Nummer der bewegten Scheibe an die Turmcodierung angefügt werden. Man sollte sich diesen Teilschritt genau ansehen. Nun ist eine durch drei Bytes codierte Anweisung abgearbeitet, und der Sequenzzeiger ist entsprechend um drei zurückzusetzen.

Die bereitgestellten Konstanten und der Arbeitsbereich wurden bereits ausführlich besprochen.

```
023E' 082B 0007 000E SEITL BTST    #7,ZURSE(A3)
0244' 6732           BEQ.S LINKS
0246' 446B 000E      RECHZ NEG.W   ZURSE(A3)
024A' 3C2B 000E            MOVE.W  ZURSE(A3),D6
024E' CCFC 0096            MULU    #150,D6
0252' 43D3           LOPRE LEA     (A3),A1
0254' 7202                 MOVEQ   #2,D1
0256' 6100 FE06            BSR     KASTN
025A' 066B 0001 0004       ADDI.W  #1,XVOLI(A3)
0260' 43D3                 LEA     (A3),A1
0262' 7205                 MOVEQ   #5,D1
0264' 6100 FDF8            BSR     KASTN
0268' 3A2B FFFE            MOVE.W  VERZ(A3),D5
026C' 4E71           WARTR NOP
026E' 51CD FFFC            DBRA    D5,WARTR
0272' 51CE FFDE            DBRA    D6,LOPRE
0276' 4E75                 RTS
0278' 3C2B 000E      LINKS MOVE.W  ZURSE(A3),D6
027C' CCFC 0096            MULU    #150,D6
0280' 43D3           LOPLI LEA     (A3),A1
0282' 7202                 MOVEQ   #2,D1
0284' 6100 FDD8            BSR     KASTN
0288' 046B 0001 0004       SUBI.W  #1,XVOLI(A3)
028E' 43D3                 LEA     (A3),A1
0290' 7205                 MOVEQ   #5,D1
0292' 6100 FDCA            BSR     KASTN
0296' 3A2B FFFE            MOVE.W  VERZ(A3),D5
029A' 4E71           WARTL NOP
029C' 51CD FFFC            DBRA    D5,WARTL
02A0' 51CE FFDE            DBRA    D6,LOPLI
02A4' 4E75                 RTS

02A6' 3C2B 0010      SENKE MOVE.W  NACHU(A3),D6
02AA' CCFC 000A            MULU    #10,D6
02AE' 43D3           LOPSE LEA     (A3),A1
02B0' 7202                 MOVEQ   #2,D1
02B2' 6100 FDAA            BSR     KASTN
02B6' 066B 0001 0006       ADDI.W  #1,YVOOB(A3)
02BC' 43D3                 LEA     (A3),A1
02BE' 7205                 MOVEQ   #5,D1
02C0' 6100 FD9C            BSR     KASTN
02C4' 3A2B FFFE            MOVE.W  VERZ(A3),D5
02C8' 4E71           WARTS NOP
```

```
02CA'  51CD FFFC            DBRA     D5,WARTS
02CE'  51CE FFDE            DBRA     D6,LOPSE
02D2'  4E75                 RTS
02D4'  6100 FDBC      BUCHE BSR      HOLTV
02D8'  0412 0001            SUBI.B   #1,(A2)
02DC'  6100 FDBA            BSR      HOLTB
02E0'  0612 0001            ADDI.B   #1,(A2)
02E4'  1012                 MOVE.B   (A2),D0
02E6'  4880                 EXT.W    D0
02E8'  322B 0008            MOVE.W   SCHNR(A3),D1
02EC'  1581 0000            MOVE.B   D1,0(A2,D0.W)
02F0'  49EC FFFD            LEA      -3(A4),A4
02F4'  4E75                 RTS

02F6'  01B4           BODEN DC.W     436
02F8'  0004                 DC.W     4
02FA'  0006                 DC.W     6
02FC'  00B0                 DC.W     176
02FE' =0002           EBER  DS.W     1       * ANZAL
0300' =0002                 DS.W     1       * VONTU
0302' =0002                 DS.W     1       * BISTU
0304' =0002                 DS.W     1       * VERZ
0306' =0002           SCHDF DS.W     1       * BREIT
0308' =0002                 DS.W     1       * DICKE
030A' =0002                 DS.W     1       * XVOLI
030C' =0002                 DS.W     1       * YVOOB
030E' =0002                 DS.W     1       * SCHNR
0310' =0002                 DS.W     1       * MAXHO
0312' =0002                 DS.W     1       * HEBHO
0314' =0002                 DS.W     1       * ZURSE
0316' =0002                 DS.W     1       * NACHU
0318' =0010           TURM1 DS.B     16
0328' =0010           TURM2 DS.B     16
0338' =0010           TURM3 DS.B     16
0348' =0004           REIHE DS.B     4       * FFFFFFFF
034C' =0001                 DS.B     1       * ZIEL
034D' =0001                 DS.B     1       * QUELLE
034E' =0001                 DS.B     1   * TOS = ANZAL
034F' =0001                 DS.B     1
0350' =018C                 DS.L     99
```

P.28.1 Die Türme von Hanoi

28.7 Vom Status des Programms

Die Überschrift wirft die Frage auf, in welcher Weise das Programm vom Betriebssystem der Abarbeitung zugeführt wird. Bei einem BASIC-Programm ist das für den Anwender äußerst einfach: Er formuliert sein Programm auf einer ziemlich hohen Problemebene und kümmert sich um die Niederungen des internen Ablaufs im allgemeinen überhaupt nicht.

Die einwandfreie Übergabe eines Maschinenprogramms an ein Betriebssystem zwecks Abarbeitung in einem Multi-User-System ist schwierig; im letzten Drittel des Buches wird darauf näher eingegangen.

Das Einbinden eines Maschinenprogramms in ein BASIC-Hostprogramm ist eine dritte und recht einfache Möglichkeit. Erinnert sei an den Aufruf mit CALL, bei dem übrigens – von uns bisher nicht praktiziert – auch noch Parameter an die Register übergeben werden können (siehe QL-Handbuch). Vor dem Rücksprung nach BASIC (mit RTS) war D0 auf Null zu setzen. Daß das Eratosthenes-Programm ohne Erfüllung der strengen QDOS-Konventionen als Code-File einfach mit exec gestartet werden konnte – und lief! –, kann nur dankbar hingenommen werden.

Weshalb kann nun dieser einfache Weg nicht auch für das Hanoi-Programm gewählt werden? Er kann, aber dann wäre der Spielablauf ein für allemal festgelegt, es sei denn, man wollte jedesmal den Quelltext (geringfügig) verändern und neu assemblieren. Eine direkte Eingabe in ein Maschinenprogramm ist im Moment auch noch schwer zu verstehen.

Wir wählen also wieder das Einbetten in ein Hostprogramm, brauchen dazu aber nicht mit Frust und Verdruß 766 Bytes Objektcode einzutippen, sondern laden mit LBYTES mdv2_TvHanoi_c,anfad den vom Assembler erzeugten Maschinencode in das BASIC-Programm ein. Hier zeigt sich wieder einmal die Effektivität der speicherplatzunabhängigen Adressierung. Manche Z80-Assembler liefern ebenfalls einen relokatiblen Code, doch der ist nicht ohne Anpassung lauffähig; er muß von einem LINKER/LOADER geladen werden und wird dabei an den konkreten Ort angepaßt. Der positionsunabhängige M68000-Code kann unmittelbar vom Band geladen und dann gestartet werden. Dabei ist es aber wichtig, mit RESPR ausreichend Platz – man denke an die wachsende Sequenz – zu reservieren; sonst könnte der Arbeitsbereich in den «Jobkontrollblock» hineinwachsen, der hinter dem BASIC-Programm im Speicher angeordnet ist. Er gehört zu dem Arbeitsbereich des Betriebssystems, in dem es Daten findet, die unmittelbar für den Ablauf des BASIC-Programms wichtig sind.

Das EinPOKEn der Aufgabenparameter wird unmittelbar verständlich sein; vielleicht aber prüfen Sie noch nach, wo die Eingaben im Objektcode ankommen.

```
100 CLS: PRINT"Tuerme von Hanoi": PRINT
110 anfad = RESPR(2000)
120 LBYTES mdv2_TvHanoi_c,anfad
130 INPUT"Anzahl Scheiben ";anzdi
140         POKE_W anfad+766,anzdi
150 INPUT"Von Turm          ";vturm
160         POKE_W anfad+768,vturm
170 INPUT"Nach Turm         ";nturm
180         POKE_W anfad+770,nturm
190 INPUT"Verzoegerung      ";verz
200         POKE_W anfad+772,verz
210 CALL anfad
220 GO TO 130
```

P.28.2 Das BASIC-Hostprogramm

28.8 Manöverkritik

Beim Entfernen einer Scheibe von ihrem bisherigen Platz läßt sie einen schmalen Strich stehen. Beim Betrachten mancher neu aufgebauten Türmchen wird man sich fragen, ob denn nun eigentlich Hanoi oder Pisa der Ort der Handlung sei. Vor allem beim NDR-Computer ist (später) ein – salopp dahingesagt – kurzfristig vorübergehender Blackout zu beobachten: Eine bewegte Scheibe scheint manchmal teilweise zu verschwinden. Die Scheiben werden stets auf hmax gehoben, auch wenn gar nicht über, von oder zu dem höchsten Turm transportiert wird.

Die folgende Aufgabe gilt im Vorgriff auch schon für die NDR-Freunde:

A.28.2:

a) Woher rühren die stehenbleibenden Striche, wie läßt sich ihr Auftreten vermeiden?

b) Kann einmal ein Schiefer Turm von Hanoi instabil werden und umzukippen drohen? Abhilfe?

c) Wie läßt sich der stroboskopische Blackout vermeiden?
d) Wie kann man unnötige Höhen vermeiden und dadurch die Ehrfurcht unbedarfter Betrachter vor dem Computer erhöhen?

28.9 Die Türme von Hanoi auf dem NDR-Computer

Nachdem P.28.1 ausführlich besprochen wurde, darf die Version für den NDR-Computer lose als Quelltext angereiht werden. Wir gehen nur auf wesentliche Abweichungen ein:

SCREEN: Daß der NDR-Bildschirm von einer Subroutine gelöscht wird, ist bereits bekannt. Zu ergänzen ist, daß grafisch dargestellte Bewegungsabläufe von der Autofliptechnik unterstützt werden: Vier Speicherbereiche können wechselweise auf dem Bildschirm ausgegeben werden. Während das eine Bild auf dem Bildschirm zu sehen ist, kann z.B. auf einer anderen «Seite» ein fertiges Bild gespeichert sein (oder aufgebaut werden), das durch einen «Flip» in kürzester Zeit zur Ausgabe gebracht werden kann; der in Abschnitt 28.8 erwähnte intermittierende Blackout kann so vermieden werden. Wir haben auf diese Möglichkeit einfachheitshalber verzichtet, ja sie stört uns sogar, so daß wir sie mit @SETFLIP abschalten. D0 = 0 setzt die Umschaltrate auf Null; D1 = 1 bestimmt die richtige Bildseite. @FIRSTTIME initialisiert die Sachildkrötengrafik, setzt die Turtle in die Mitte des Bildschirms und richtet sie nach oben.

KASTN zeichnet das Rechteck so: Zuerst wird der Schreibstift der Turtle angehoben, d.h., er hinterläßt keine Spur. Die vorgeschriebene X-Position wird aus der Rechteckdefinition nach D0 übernommen, die Y-Position aus 210 − (6(A1)) berechnet, worin 210 den Umstand korrigiert, daß Y beim NDR-Computer vom unteren Bildschirmrand an zählt, beim QL hingegen vom oberen. Nun wird der Schreibstift AUF X,Y positioniert und dann zum Zeichnen gesenkt. Die Breite des Rechtecks wird nach D0 kopiert, und die Schildkröte @SCHREITEt nun entsprechend viele Pixel in der angegebenen Richtung, also nach oben. Der Drehwinkel der Turtle wird im mathematischen Sinn angegeben; einer Rechtsdrehung entspricht also ein negativer Winkel. So wird also ein Rechteck durch zweimaligen Aufruf des Unterprogramms «Schreite um die Höhe, drehe um 90° nach rechts, schreite um die Breite geradeaus, drehe um 90° nach rechts» gezeichnet. Da diese Subroutine D0 und D1 verwendet, werden diese beiden auf den A7-Stapel gerettet. Die Aufgabenparameter werden, wie gleich zu sehen sein wird, sehr komfortabel

```
* TURM VON HANOI

ANZAHL  EQU     -8
VONTU   EQU     -6
BISTU   EQU     -4
VERZ    EQU     -2
* A3 -> SCHDF
BREIT   EQU     0
DICKE   EQU     2
XVOLI   EQU     4
YVOOB   EQU     6
SCHNR   EQU     8
MAXHO   EQU     10
HEBHO   EQU     12
ZURSE   EQU     14
NACHU   EQU     16

START:  BSR     INIT
        BSR     EINLES
        BSR     TURM0
TRIAL:  CMPI.B  #-1,(A4)
        BEQ.S   GRUPRO
        BSR     ANALYS
        BRA.S   TRIAL
GRUPRO: RTS

INIT:   BSR     SCREEN
        LEA     SCHDF(PC),A3
        LEA     REIHE(PC),A4
        BSR     LOESCH
        LEA     BODEN(PC),A1
        BSR     KASTN
        MOVE.W  #-1,(A4)+
        MOVE.W  #-1,(A4)+
        RTS

SCREEN: JSR     @CLRSCREEN.S
        MOVEQ   #0,D0
        MOVEQ   #1,D1
        JSR     @SETFLIP.S
        JSR     @FIRSTTIME.S
        RTS

LOESCH: LEA     (A3),A0
        MOVEQ   #31,D0
LOPLOE: CLR.L   (A0)+
        DBRA    D0,LOPLOE
        MOVE.W  #9,DICKE(A3)
        RTS

KASTN:  JSR     @HEBE.S
        MOVE.W  4(A1),D1
        MOVE.W  #210,D2
        SUB.W   6(A1),D2
        JSR     @AUFXY.S
        MOVEQ   #90,D0
        JSR     @AUFK.S
        JSR     @SENKE.S
        BSR     KAST2
KAST2:  MOVE.W  2(A1),D0
        JSR     @SCHREITE.S
        MOVEQ   #-90,D0
        JSR     @DREHE.S
        MOVE.W  (A1),D0
        JSR     @SCHREITE.S
        MOVEQ   #-90,D0
        JSR     @DREHE.S
        RTS

EINLES: LEA     FRAGE1(PC),A0
        BSR     EINGEB
        MOVE.W  D0,ANZAHL(A3)
        MOVE.B  D0,2(A4)
```

```
        LEA     FRAGE2(PC),A0
        BSR     EINGEB
        MOVE.W  D0,VONTU(A3)
        MOVE.B  D0,1(A4)
        LEA     FRAGE3(PC),A0
        BSR     EINGEB
        MOVE.W  D0,BISTU(A3)
        MOVE.B  D0,(A4)
        LEA     2(A4),A4
        LEA     FRAGE4(PC),A0
        BSR     EINGEB
        MOVE.W  D0,VERZ(A3)
        BSR     HOLTV
        MOVE.B  (A4),D0
        EXT.W   D0
        MOVE.B  D0,(A2)+
LEINT0: MOVE.B  D0,(A2)+
        DBRA    D0,LEINT0
        RTS

EINGEB: LEA     WRITPA(PC),A1
        MOVEM.W (A1),D0-D2
        JSR     @WRITE.S
        LEA     BUFFER(PC),A0
        LEA     READPA(PC),A1
        MOVEM.W (A1),D0-D3
        JSR     @READ.S
        LEA     BUFFER(PC),A0
        JSR     @WERT.S
        RTS

WRITPA: DC.L    $00110002
        DC.W    $00F0
READPA: DC.L    $00110030
        DC.L    $00F00006
FRAGE1: DC.B    'Anza'
        DC.B    'hl ',0
FRAGE2: DC.B    'von '
        DC.B    '   ',0
FRAGE3: DC.B    'bis '
        DC.B    '   ',0
FRAGE4: DC.B    'Verz'
        DC.B    'g. ',0
BUFFER: DS.B    8

HOLTV:  MOVE.W  VONTU(A3),D0
        BRA.S   HOLT2
HOLTB:  MOVE.W  BISTU(A3),D0
HOLT2:  ASL.W   #4,D0
        LEA     2(A3,D0.W),A2
        RTS

TURM0:  BSR     HOLTV
        LEA     1(A2),A2
        MOVE.B  (A2),D0
        BEQ     EXTURM
        EXT.W   D0
        MULU    #10,D0
        MOVE.W  D0,BREIT(A3)
        MOVE.W  VONTU(A3),D1
        MULU    #150,D1
        SUBI.W  #75,D1
        LSR.W   #1,D0
        SUB.W   D0,D1
        MOVE.W  D1,XVOLI(A3)
        MOVE.W  #170,YVOOB(A3)
        LEA     (A3),A1
LOPT0:  MOVEM.W D0-D2,-(A7)
        BSR     KASTN
        MOVEM.W (A7)+,D0-D2
        SUBI.W  #10,BREIT(A3)
        BEQ.S   EXTURM
        ADDI.W  #5,XVOLI(A3)
        SUBI.W  #10,YVOOB(A3)
        BRA.S   LOPT0
EXTURM: RTS
```

```
ANALYS: CMPI.B  #1,(A4)
        BEQ.S   ZIEHEN
REKURS: MOVE.B  (A4),D0
        SUBQ.B  #1,D0
        MOVE.B  D0,(A4)
        MOVE.B  D0,6(A4)
        MOVE.B  -1(A4),2(A4)
        MOVE.B  -1(A4),5(A4)
        MOVE.B  -2(A4),1(A4)
        MOVEQ   #6,D0
        SUB.B   -1(A4),D0
        SUB.B   -2(A4),D0
        MOVE.B  D0,4(A4)
        MOVE.B  D0,-1(A4)
        MOVE.B  #1,3(A4)
        LEA     6(A4),A4
        RTS

ZIEHEN: BSR     SCHDAT
        BSR     KASPAR
        BSR     HMAXDD
        BSR     BEWEGE
        RTS

SCHDAT: MOVE.B  -1(A4),D0
        EXT.W   D0
        MOVE.W  D0,VONTU(A3)
        MOVE.B  -2(A4),D0
        EXT.W   D0
        MOVE.W  D0,BISTU(A3)
        BSR     HOLTV
        MOVE.B  (A2),D0
        EXT.W   D0
        MOVE.B  0(A2,D0.W),D1
        EXT.W   D1
        MOVE.W  D1,SCHNR(A3)
        RTS

KASPAR: MOVE.W  SCHNR(A3),D0
        MULU    #10,D0
        MOVE.W  D0,BREIT(A3)
        MOVE.W  VONTU(A3),D1
        MULU    #150,D1
        SUBI.W  #75,D1
        LSR.W   #1,D0
        SUB.W   D0,D1
        MOVE.W  D1,XVOLI(A3)
        BSR     HOLTV
        MOVE.B  (A2),D0
        EXT.W   D0
        MULU    #10,D0
        MOVE.W  #180,D1
        SUB.W   D0,D1
        MOVE.W  D1,YVOOB(A3)
        RTS

HMAXDD: MOVE.B  18(A3),D0
        CMP.B   34(A3),D0
        BCC.S   HMAX1
        MOVE.B  34(A3),D0
HMAX1:  CMP.B   50(A3),D0
        BCC.S   HMAX2
        MOVE.B  50(A3),D0
HMAX2:  EXT.W   D0
        MOVE.W  D0,MAXHO(A3)
        BSR     HOLTV
        MOVE.W  MAXHO(A3),D0
        SUB.B   (A2),D0
        ADDQ.B  #1,D0
        EXT.W   D0
        MOVE.W  D0,HEBHO(A3)
        BSR     HOLTB
        MOVE.W  MAXHO(A3),D0
        SUB.B   (A2),D0
        EXT.W   D0
```

```
        MOVE.W  D0,NACHU(A3)
        MOVE.W  VONTU(A3),D0
        SUB.W   BISTU(A3),D0
        MOVE.W  D0,ZURSE(A3)
        RTS

BEWEGE: BSR     HEBEN
        BSR     SEITLI
        BSR     SENKE
        BSR     BUCHE
        RTS

HEBEN:  MOVE.W  HEBHO(A3),D6
        MULU    #10,D6
LOPHEB: JSR     @ERAPEN.S
        BSR     KASTN
        JSR     @SETPEN.S
        SUBI.W  #1,YVOOB(A3)
        BSR     KASTN
        MOVE.W  VERZ(A3),D5
WARTEH: DBRA    D5,WARTEH
        DBRA    D6,LOPHEB
        RTS

SEITLI: BTST    #7,ZURSE(A3)
        BEQ.S   LINKS
RECHTS: NEG.W   ZURSE(A3)
        MOVE.W  ZURSE(A3),D6
        MULU    #150,D6
LOPRE:  JSR     @ERAPEN.S
        BSR     KASTN
        JSR     @SETPEN.S
        ADDI.W  #1,XVOLI(A3)
        BSR     KASTN
        MOVE    VERZ(A3),D5
WARTER: DBRA    D5,WARTER
        DBRA    D6,LOPRE
        RTS
LINKS:  MOVE.W  ZURSE(A3),D6
        MULU    #150,D6
LOPLI:  JSR     @ERAPEN.S
        BSR     KASTN
        JSR     @SETPEN.S
        SUBI.W  #1,XVOLI(A3)
        BSR     KASTN
        MOVE    VERZ(A3),D5
WARTEL: DBRA    D5,WARTEL
        DBRA    D6,LOPLI
        RTS
```

P.28.3 Turm von Hanoi für den NDR-Computer

nach D0 eingegeben. In EINLES gelangen sie an dieselben Stellen wie im QL-Programm.

EINGEB erwartet zunächst eine Ausrichtung des Pointers A0 auf den auszugebenden Text; @WRITE ist ja inzwischen gut bekannt. Die Eingabe wird von @READ in den Puffer gelesen, wo abschließend @WERT den vom eingelesenen String dargestellten Zahlenwert ermittelt und in D0 zurückgibt.

Die folgenden Routinen stimmen nun absolut mit denen aus P.28.1 überein; lediglich Farbangaben entfallen, und die Motorola-Norm «(PC)» muß erfüllt werden; das erwartete der Metacomco-Assembler ohne ORG-Angabe anders.

Bei Bewegungen einer Scheibe auf dem Bildschirm ist nun nicht wie beim QL für das Löschen die Hintergrundfarbe (in D1) und für das Hinzeichnen eine andere Farbe zu definieren, sondern mal trägt die Turtle einen Löschstift (ERAPEN), dann wieder einen Schreibstift (SETPEN). Beachten Sie, daß der R.-D.-Klein-Assembler eine Warte-auf-der-Stelle-Schleife richtig codiert.

```
SENKE:  MOVE.W  NACHU(A3),D6
        MULU    #10,D6
LOPSEN: JSR     @ERAPEN.S
        BSR     KASTN
        JSR     @SETPEN.S
        ADDI.W  #1,YVOOB(A3)
        BSR     KASTN
        MOVE.W  VERZ(A3),D5
WARTES: DBRA    D5,WARTES
        DBRA    D6,LOPSEN
        RTS

BUCHE:  BSR     HOLTV
        SUBI.B  #1,(A2)
        BSR     HOLTB
        ADDI.B  #1,(A2)
        MOVE.B  (A2),D0
        EXT.W   D0
        MOVE.W  SCHNR(A3),D1
        MOVE.B  D1,0(A2,D0.W)
        LEA     -3(A4),A4
        RTS

BODEN:  DC.W    436
        DC.W    4
        DC.W    6
        DC.W    176

EBER:   DS.W    1       * ANZAHL
        DS.W    1       * VONTU
        DS.W    1       * BISTU
        DS.W    1       * VERZ
SCHDF:  DS.W    1       * BREIT
        DS.W    1       * DICKE
        DS.W    1       * XVOLI
        DS.W    1       * YVOOB
BEWPA:  DS.W    1       * SCHNR
        DS.W    1       * MAXHO
        DS.W    1       * HEBHO
        DS.W    1       * ZURSE
        DS.W    1       * NACHU
TURM1:  DS.B    16
TURM2:  DS.B    16
TURM3:  DS.B    16
REIHE:  DS.B    4       * FFFFFFFF
        DS.B    1       * ZIEL
        DS.B    1       * QUELLE
        DS.B    1   * TOS = ANZAHL
        DS.L    99

        END
```

P.28.3 Turm von Hanoi für den NDR-Computer (IV)

29
Disassembler

Ein Disassembler kehrt das Assemblieren um und übersetzt ein Programm aus der Maschinensprache in die Assemblersprache zurück. Da das Maschinenprogramm lediglich das Endresultat des Assemblierens ist, enthält es keine Information darüber, in welcher Form der Quelltext vorlag, ob dort z.B. ein Speicherplatz absolut adressiert wurde, ob durch eine EQU-Direktive, durch ein Label oder sogar als Label+Zahl. Der Disassembler findet lediglich Bitmuster im Speicher vor, weiß aber nicht, ob sie dezimal oder hexadezimal vorgegeben waren. Findet ein Disassembler ein nicht disassemblierbares Datum – ein Datenwort, dem kein Mnemonic entspricht –, meldet er einen Fehler. Er erkennt nämlich in aller Regel nicht, daß er eine Tabelle,

```
022222                          ORG     $22222
022222  600A        MARKE:      BRA.S   START
022224  F234                    DC.W    $F234
022226  556E 7369   TEXT:       DC.B    'Unsinn!',0
02222A  6E6E 2100
02222E  41FA FFF4   START:      LEA     START-10(PC),A0
022232  6100 FFF0               BSR     MARKE
022236  4E75                    RTS

022222  600A        BRA.S       $0002222E
022224  F234                    <<< FEHLER <<< 0005FFF2
022226  556E 7369   SUBQ.W      #2,$7369(A6)
02222A  6E6E        BGT.S       $0002229A
02222C  2100        MOVE.L      D0,-(A0)
02222E  41FA FFF4   LEA         $FFF4(PC),A0   =00022224
022232  6100 FFEE   BSR         $00022222
022236  4E75        RTS
```

Bild 29.1 Assemblerprotokoll (oben), Disassemblerauszug (unten)

Distanz-, Adreß- oder Texttabelle erreicht hat. Sinn und Bedeutung erkennt (fast) nur der Mensch, nicht ein Programm oder eine Maschine.

Den Unterschied zwischen Quelltext und Disassemblerauszug zeigt Bild 29.1.

29.1 Entwurf eines Disassemblers

Offenbar muß ein Disassembler so arbeiten: Er erwartet zunächst die Eingabe zweier Adressen, VON der BIS zu der er das Maschinenprogramm zurückübersetzen soll. Dann stellt er im AusgabeBEReich die erste (VON)-Adresse als String bereit und holt aus dieser Adresse ein Datenwort, den ersten zu analysierenden Opcode. Er wird mit einem kleinen Zwischenraum an den Adreßstring angehängt. Dann beginnt die eigentliche Analyse, für die mindestens zwei Algorithmen in Betracht kommen:

Frühe Mehrfachauswahl: Man teilt das Programm nach Maßgabe des MSN des Opcodes in sechzehn parallele Zweige. Man gewinnt dadurch Zeit, denn man braucht ja nicht erst prüfen zu lassen, ob der Opcode Axxx unter 0xxx, 1xxx, 2xxx, ... zu finden ist. Der Zeitgewinn würde aber kaum ins Gewicht fallen, denn man will das Disassemblat ja sehen, entweder auf dem Bildschirm oder auf dem Papier. In der Realität bestimmen die langsamen Medien Bildschirm und Drucker das Ablauftempo.

Ein anderer Gesichtspunkt spricht dagegen: Unter den Opcodes 1xxx verbergen sich die MOVE.B-Befehle, unter 2xxx die MOVE.L- und unter 3xxx die MOVE.W-Befehle. Eine rigorose Aufsplittung in sechzehn parallele Zweige würde die einheitliche Behandlung solcher Übereinstimmungen nicht gerade nahelegen. Unser Lösungsvorschlag beruht auf der Beobachtung, daß z.B. ADDQ so codiert wird:

```
ADDQ.X: 0101 ddd0 ggmm mrrr
Darin bedeutet        00        .B,
                      01        .W,
                      10        .L.
```

Was aber, wenn hier 11 steht? Der Disassembler deutet 0101 0100 1100 0000 als SCC D0. Die Größen-(Format-)Codierung 11 ist also nicht einfach verboten, sondern definiert einen anderen Befehl. Es leuchtet ein, daß man die drei ADDQs geschlossen zurückübersetzen kann, wenn man sicherstellt, daß der Fall 0101 ddd0 11mm mrrr nicht

29.2 Speicher- und Registerdisposition

Ausgehend von der gewünschten Ausgabeform werden im Arbeitsbereich des Programms folgende Felder (siehe Symboltabelle Bild 29.2 in Abschnitt 29.12) bereitgestellt:

```
+--------------------------------------- ABER1
!                     +----------------- ABER2
!                     !           +-- ABER3
022222   600A         BRA.S       $ 0002222E
```

Dank der erfreulich großen Anzahl der im NDR-Computer frei verfügbaren Register (bis auf A5 alle) brauchen nur ganz selten Daten auf dem Stapel hinterlegt zu werden, und die Register können weitestgehend die einmal zugeordnete Funktion beibehalten:

A0: ist auf ABER1 gerichtet und leitet die Bildung der Adreß- und Operandenstrings nach dort.

A1: bedient ABER2 (Mnemonics, Formatangaben) und ABER3 (Quell- und Zieloperanden).

A2, A4 und A6: zur freien momentanen Verfügung.

A3: ist auf das jeweilige Datenwort (Opcode, Operand) im Speicher gerichtet.

A7: wie immer der Stapelzeiger. Sein anfänglicher Inhalt wird sofort in DEPA7 deponiert. Fehlermeldungen können nämlich in unterschiedlichen Unterprogrammebenen ausgelöst werden. Da dann bei einem erzwungenen Rücksprung in die Ebene0 der Stapel in Unordnung geraten bzw. über- oder unterzählt werden könnte, wird am Ende der Fehlergenerierung der Stackpointer mit seinem ursprünglichen Wert geladen. Die erwähnten Felder ABER1 bis ABER3 sind als letzte im Arbeitsbereich angeordnet. So wird vermieden, daß z.B. das BIS-Feld überschrieben wird und das Programm kein Ende findet.

29.3 Die Subroutine OPCODE (20AB0)

Anstelle einer quasi-chronologischen Besprechung des Listings folgen wir vorläufig dem Ablauf des Disassemblierens. Die eigentliche Decodierung erfolgt in einer Subroutine, deren letztes RTS somit zur Ausgabe des bereitgestellten Strings führt.

```
020078  00004AFC        TAB1:   DC.L    $00004AFC
020080  00004E70                DC.L    $00004E70
020084  00004E71                DC.L    $00004E71
020088  00004E73                DC.L    $00004E73
02008C  00004E75                DC.L    $00004E75
020090  00004E76                DC.L    $00004E76
020094  00004E77                DC.L    $00004E77
020098  00024E72                DC.L    $00024E72
02009C  494C4C4547414C          DC.B    'ILLEGAL',0
0200A3  00
0200A4  52455345542020          DC.B    'RESET  ',0
0200AB  00
0200AC  4E4F5020202020          DC.B    'NOP    ',0
0200B3  00
0200B4  52544520202020          DC.B    'RTE    ',0
0200BB  00
0200BC  52545320202020          DC.B    'RTS    ',0
0200C3  00
0200C4  54524150562020          DC.B    'TRAPV  ',0
0200CB  00
0200CC  52545220202020          DC.B    'RTR    ',0
0200D3  00
0200D4  53544F50202020          DC.B    'STOP   ',0
0200DB  00
0200DC                  * ================================
0200DC  6100 09D2       GRUP2:  BSR     OPCODE
0200E0  0243 FFBF               ANDI.W  #$FFBF,D3
0200E4  49FA 001A               LEA     TAB2(PC),A4
0200E8  7202                    MOVEQ   #2,D1
0200EA  45FA 0008               LEA     GR2W(PC),A2
0200EE  6100 09FA               BSR     SELECT
0200F2  6630                    BNE.S   GRUP3
0200F4  45FA 0F21       GR2W:   LEA     OC+01(PC),A2
0200F8  0812 0006               BTST    #6,(A2)
0200FC  6000 0B1C               BRA     CKSRCR
020100  0000003C        TAB2:   DC.L    $0000003C
020104  00000023C               DC.L    $00000023C
020108  00000A3C                DC.L    $00000A3C
02010C  4F524920202020          DC.B    'ORI    ',0
020113  00
020114  414E4449202020          DC.B    'ANDI   ',0
02011B  00
02011C  454F5249202020          DC.B    'EORI   ',0
020123  00
020124
```

Die erste wesentliche Sub-Subroutine trägt dem Codierungs-Grundschema des M68000 Rechnung:

xxxx rrr mmm mmm rrr

Wir nennen die auf das MSN folgenden Tripel b2, b3, b4 und b5. OPCODE holt den Opcode aus dem Speicher, legt ihn im Feld OC ab, errechnet die Werte der vier Tripel und legt sie hinter OC ab. Diese Aufbereitung findet immer statt, auch wenn sie im Einzelfall nicht nötig sein sollte. Der Opcode verbleibt zunächst auch in D3.

29.4 Die Subroutine SELECT (20AEA)

Überlegungsgemäß beginnt unser Disassembler bei den Opcodes mit der höchsten Anzahl, also sechzehn, fest definierter Bits. Die n = 8 Opcodes dieser Gruppe 1 und die zugehörigen Mnemonics werden in einer Tabelle von n Langwörtern und n (einschließlich Schlußnull) acht Zeichen langen Strings bereitgestellt. Die LSWs dieser Langwörter enthalten die Opcodes, die MSWs eine Adreßdistanz zur jeweiligen Verarbeitungsroutine, auf die wir noch zurückkommen.

SELECT vergleicht den in D3 stehenden zu decodierenden Opcode mit den in der Tabelle aufgezeichneten und lädt bei Übereinstimmung A4 mit der Adresse der zugehörigen Aufbereitungsroutine. Vor dem Aufruf von SELECT müssen außer dem Opcode in D3 noch der Tabellenanfang in A4, die um 1 verminderte Anzahl der Tabelleneinträge, also n−1, in D1 und der Anfang der jeweiligen Liste der Aufbereitungsroutinen in A2 bereitgestellt werden.

SELECT hat zwei Ausgänge: Wurde der Opcode gefunden, geht es mit JMP (A4) nach Ausgleich des Stapelzeigers A7 sofort zur Aufbereitung, anderenfalls mit RTS zurück zur aufrufenden Stelle und von dort zum Vergleich mit der nächsten Befehlsgruppe.

SELECT kopiert zunächst n−1 in den Schleifenzähler D4. Nun werden der Reihe nach die n Langwörter aus der Tabelle nach D5 geholt und ihr LSW (der Opcode) mit dem in D3 stehenden verglichen. Bei erfolgloser Suche wird der SELLOP-Loop mit gelöschtem Z-Flag durch RTS verlassen.

Falls der fragliche Opcode in der Tabelle gefunden wurde, wird die immer noch in D1 stehende Anzahl n−1 nach D0 transferiert und dort verdoppelt. Nach Subtraktion des Schleifenzählers wird das Ergebnis vervierfacht. Das erste LEA richtet nun, wie man leicht an einem konkreten Beispiel nachvoll-

```
020124  3613            GRUP3:  MOVE.W  (A3),D3
020126  0243 FFF0              ANDI    #$FFF0,D3
02012A  0C43 4E40              CMPI    #$4E40,D3
02012E  6618                   BNE.S   GRUP4
020130  43FA 0F22              LEA     ABER2(PC),A1
020134  22BC 54524150          MOVE.L  #'TRAP',(A1)
02013A  43FA 0F20              LEA     ABER3(PC),A1
02013E  3013                   MOVE.W  (A3),D0
020140  6100 0A50              BSR     NIBMGS
020144  12C0                   MOVE.B  D0,(A1)+
020146  4E75                   RTS
020148                  * ==========================================
020148  6100 0966      GRUP4:  BSR     OPCODE
02014C  6100 09D2              BSR     BSRN
020150  0243 F000              ANDI.W  #$F000,D3
020154  0C43 4000              CMPI.W  #$4000,D3
020158  6600 008E              BNE     GRUP5
02015C  363A 0EB8              MOVE.W  OC(PC),D3
020160  0243 FFF8              ANDI.W  #$FFF8,D3
020164  49FA 002E              LEA     TAB4(PC),A4
020168  7206                   MOVEQ   #6,D1
02016A  45FA 0008              LEA     GR4W(PC),A2
02016E  6100 097A              BSR     SELECT
020172  6674                   BNE.S   GRUP5
020174  6000 0B6E      GR4W:   BRA     DRGDIR
020178  6100 0B76              BSR     ARGDIR
02017C  6000 0CAC              BRA     KCNBWL
020180  6000 0B6E              BRA     ARGDIR
020184  6100 0B6A              BSR     ARGDIR
020188  6000 0CC2              BRA     KUSPAN
02018C  6100 0CC2              BSR     USPRAN
020190  6000 0B5A              BRA     KARDIR
020194  00004840        TAB4:   DC.L    $00004840
020198  00004880                DC.L    $00004880
02019C  000048C0                DC.L    $000048C0
0201A0  00044E50                DC.L    $00044E50
0201A4  000C4E58                DC.L    $000C4E58
0201A8  00104E60                DC.L    $00104E60
0201AC  00184E68                DC.L    $00184E68
0201B0  53574150202020          DC.B    'SWAP   ',0
0201B7  00
0201B8  4558542E572020          DC.B    'EXT.W  ',0
0201BF  00
0201C0  4558542E4C2020          DC.B    'EXT.L  ',0
0201C7  00
0201C8  4C494E4B202020          DC.B    'LINK   ',0
0201CF  00
0201D0
```

ziehen kann, A4 auf den gesuchten MNEmonik, der von MNEGNZ GaNZ nach ABER2 kopiert wird. Weil damit der Mnemonik vollständig ist, richtet MNEGNZ A1 sofort auf ABER3.

Nachdem die Adreßdistanz vom Label GR1W («Gruppe1 weiter») zur Aufbereitungsroutine in das LSW von D5 geSWAPped wurde, lädt das zweite LEA A4 mit der Adresse der zum Opcode gehörenden Routine. Für Gruppe 1 sind nur zwei derartige Aufbereitungsroutinen erforderlich:

Zu den 7 Mnemonics ILLEGAL bis RTR gehört keine weitere Angabe (Operand usw.); hier führt die Adreßdistanz 0 von GR1W nach RTS, und es geht zur Ausgabe zurück ins Hauptprogramm.

Für STOP führt die Adreßdistanz 2 nach CONBWL (Byte-, Wort- oder Langwort-CONstante). Diese Routine verdient eine gesonderte Betrachtung.

29.5 Die Subroutine CONBWL (20E2E)

Nachdem das grundlegende Datenformat des M68000 das Wort ist, darf man Byte und Langwort als Ausnahmen von dieser Regel ansehen. Findet nun CONBWL hinter dem Mnemonic .B oder .L, so sind Konstanten dieser Formate «Mit Gitter und Schlange» (MGS = #$) anzufügen, anderenfalls eine Wortkonstante MGS.

Um das herauszufinden, sucht CONBWL zunächst nach dem Punkt, der das Format vom Mnemonic trennt. Wurde kein Punkt gefunden, so ist das gleichbedeutend mit «keine Formatangabe, also Wort», und es geht bei WGEFUN: (Wort GEFUNden) weiter: WRTMGS hängt ein Wort mit Gitter und Schlange an. Wird aber ein Punkt gefunden, geht es bei einem folgenden B weiter bei BYTMGS und bei einem L weiter bei LAWMGS (LAngWort MGS). «Weder B noch L» ist gleichbedeutend mit .W und führt ebenfalls nach WRTMGS.

WRTMGS ruft zunächst die Subroutine NXTWRT auf; sie holt das nächste Wort aus dem Speicher nach D0 und schreibt dies auch in das Feld NW ein. Dies zum Befehl gehörende zweite Datenwort muß auch in ABER1 als Objektcode-String eingetragen werden. A0 zeigt auf das nächste hier zu beschreibende Feld, in das zunächst ein Space-Code eingetragen wird. Daß nun D0 noch einmal aus (A3) geladen wird, mag überflüssig erscheinen. So aber kann WORTAN auch ohne Vorspann aufgerufen werden. Die Umwandlung des Worts in einen String und seine Ablage vom aktuellen A0-Stand an bewirkt wieder @PRINT4X. Diese Systemroutine kennzeichnet

```
0201D0  554E4C4B202020          DC.B    'UNLK   ',0
0201D7  00
0201D8  4D4F5645202020          DC.B    'MOVE   ',0
0201DF  00
0201E0  4D4F5645202020          DC.B    'MOVE   ',0
0201E7  00
0201E8
0201E8  6100 08C6       GRUP5:  BSR     OPCODE
0201EC  6100 0932               BSR     B5RN
0201F0  0243 FFC0               ANDI.W  #$FFC0,D3
0201F4  49FA 004C               LEA     TAB5(PC),A4
0201F8  7213                    MOVEQ   #19,D1
0201FA  45FA 000A               LEA     GR5W(PC),A2
0201FE  6100 08EA               BSR     SELECT
020202  6600 012E               BNE     GRUP5A
020206  6100 0BE4       GR5W:   BSR     BITNRZ
02020A  6000 0A58               BRA     KEA7FD
02020E  6100 0BDC               BSR     BITNRZ
020212  6000 0A30               BRA     KEA1FD
020216  6100 0C4A               BSR     SR6RAN
02021A  6000 0A52               BRA     KEAFFD
02021E  6100 0A52               BSR     EA0FFD
020222  6000 0C48               BRA     KCCRAN
020226  6100 0A4A               BSR     EA0FFD
02022A  6000 0C32               BRA     KSRGAN
02022E  6000 0A18               BRA     EA01FD
020232  6000 0A2A               BRA     EA07F4
020236  6000 0A10               BRA     EA01FD
02023A  6000 0A22               BRA     EA07F4
02023E  6000 09FE               BRA     EA01FC
020242  00000800        TAB5:   DC.L    $00000800
020246  00080840                DC.L    $00080840
02024A  00080880                DC.L    $00080880
02024E  000808C0                DC.L    $000808C0
020252  001040C0                DC.L    $001040C0
020256  001844C0                DC.L    $001844C0
02025A  002046C0                DC.L    $002046C0
02025E  00284800                DC.L    $00284800
020262  002C4840                DC.L    $002C4840
020266  00304AC0                DC.L    $00304AC0
02026A  00344E80                DC.L    $00344E80
02026E  00344EC0                DC.L    $00344EC0
020272  0038E0C0                DC.L    $0038E0C0
020276  0038E1C0                DC.L    $0038E1C0
02027A  0038E2C0                DC.L    $0038E2C0
02027E  0038E3C0                DC.L    $0038E3C0
020282  0038E4C0                DC.L    $0038E4C0
020286
```

aber das Ende durch das Byte 00, was zu unliebsamen Störungen, z.B. zur vorzeitigen Beendigung der Ausgabe, führen kann. Deshalb wird diese Stelle ohne Postinkrement mit einem Space überschrieben.

WRTMGS fügt dann Gitter und Schlange an und verzweigt nach WRT2AN, wo der unveränderte Inhalt von D0 als String in ABER3 abgelegt wird. Hierzu vertauschen A0 und A1 vorübergehend ihre Funktion. WRT2AN endet mit RTS, führt also in das Hauptprogramm zurück. Dort folgt auf DECODE AUSGEB.

29.6 Die Subroutine AUSGEB (20BAC)

Beim Programmstart wurde in das Feld JOB eine 3-bit-Zahl eingeschrieben. Bit0 = 1 lenkt die Ausgabe zum Bildschirm, Bit1 = 1 zum Drucker. Bit2 wählt zwischen Disassemblieren und Ziehen eines Hexdumps. Im einzelnen:

JOB = 0 oder 4: zurück ins Betriebssystem
= 1 Hexdump zum Bildschirm
= 2 Hexdump zum Drucker
= 3 Hexdump nach Drucker und Bildschirm
= 5 Disassemblieren auf Bildschirm
= 6 Disassemblieren auf Drucker
= 7 Disassemblieren auf Bildschirm und Drucker.

Dementsprechend prüft AUSGEB anfangs, welche Bits gesetzt sind, und leitet danach weiter zu DISAUS, PRIAUS oder DISAUS und PRIAUS.

29.7 Die Befehle der Gruppe 2 (2000DC)

Folgende Opcodes sollen erkannt werden:

```
0000 0010 0011 1100 = ANDI to CCR
0000 0010 0111 1100 = ANDI to SR
0000 1010 0011 1100 = EORI to CCR
0000 1010 0111 1100 = EORI to SR
0000 0000 0011 1100 = ORI  to CCR
0000 0000 0111 1100 = ORI  to SR
1111 1111 1011 1111 = $FFBF
```

```
020286
020286  0038E5C0                DC.L    $0038E5C0
02028A  0038E6C0                DC.L    $0038E6C0
02028E  0038E7C0                DC.L    $0038E7C0
020292  42545354202020         DC.B    'BTST   ',0
020299  00
02029A  42434847202020         DC.B    'BCHG   ',0
0202A1  00
0202A2  42434C52202020         DC.B    'BCLR   ',0
0202A9  00
0202AA  42534554202020         DC.B    'BSET   ',0
0202B1  00
0202B2  4D4F5645202020         DC.B    'MOVE   ',0
0202B9  00
0202BA  4D4F5645202020         DC.B    'MOVE   ',0
0202C1  00
0202C2  4D4F5645202020         DC.B    'MOVE   ',0
0202C9  00
0202CA  4E424344202020         DC.B    'NBCD   ',0
0202D1  00
0202D2  50454120202020         DC.B    'PEA    ',0
0202D9  00
0202DA  54415320202020         DC.B    'TAS    ',0
0202E1  00
0202E2  4A535220202020         DC.B    'JSR    ',0
0202E9  00
0202EA  4A4D5020202020         DC.B    'JMP    ',0
0202F1  00
0202F2  4153522E572020         DC.B    'ASR.W  ',0
0202F9  00
0202FA  41534C2E572020         DC.B    'ASL.W  ',0
020301  00
020302  4C53522E572020         DC.B    'LSR.W  ',0
020309  00
02030A  4C534C2E572020         DC.B    'LSL.W  ',0
020311  00
020312  524F58522E5720         DC.B    'ROXR.W ',0
020319  00
02031A  524F584C2E5720         DC.B    'ROXL.W ',0
020321  00
020322  524F522E572020         DC.B    'ROR.W  ',0
020329  00
02032A  524F4C2E572020         DC.B    'ROL.W  ',0
020331  00
020332  6100 077C       GRUP5A: BSR     OPCODE
020336  6100 07E8               BSR     B5RN
02033A  0243 FB80               ANDI.W  #$FB80,D3
02033E  0C43 4880               CMPI.W  #$4880,D3
020342
```

```
020342
020342  6658                        BNE.S   GRUP6
020344  49FA 004E                   LEA     TAB5A(PC),A4
020348  6100 0840                   BSR     MNEGNZ
02034C  082B 0006 0001              BTST    #6,01(A3)
020352  6706                        BEQ.S   NOTL
020354  137C 004C FFFE              MOVE.B  #'L',-2(A1)
02035A  6100 07E4           NOTL:   BSR     NXTWRT
02035E  49FA 0CC0                   LEA     CODLIS(PC),A4
020362  38AC FFFE                   MOVE.W  -2(A4),(A4)
020366  082B 0002 FFFE              BTST    #2,-2(A3)
02036C  670C                        BEQ.S   REGMEM
02036E  6100 08E8           MEMREG: BSR     EA07EC
020372  12FC 002C                   MOVE.B  #',',(A1)+
020376  6000 0B36                   BRA     INVLIS
02037A  45FA 0C9E           REGMEM: LEA     B4(PC),A2
02037E  0C12 0004                   CMPI.B  #4,(A2)
020382  6608                        BNE.S   ERSTIN
020384  6100 0B3E                   BSR     REGLIS
020388  6000 08A6                   BRA     KEA1F4
02038C  6100 0B20           ERSTIN: BSR     INVLIS
020390  6000 089E                   BRA     KEA1F4
020394  4D4F56454D2E57   TAB5A:  DC.B    'MOVEM.W',0
02039B  00
02039C                      *  ==============================
02039C  6100 0712           GRUP6:  BSR     OPCODE
0203A0  0243 F1F8                   ANDI.W  #$F1F8,D3
0203A4  49FA 006E                   LEA     TAB6(PC),A4
0203A8  720A                        MOVEQ   #10,D1
0203AA  45FA 000A                   LEA     GR6W(PC),A2
0203AE  6100 073A                   BSR     SELECT
0203B2  6600 00E4                   BNE     GRUP7
0203B6  6100 0768           GR6W:   BSR     B5RN
0203BA  6100 0968                   BSR     AIDIST
0203BE  6018                        BRA.S   SUP2
0203C0  6100 0754                   BSR     B2RN
0203C4  6100 091E                   BSR     DRGDIR
0203C8  6100 0756                   BSR     B5RN
0203CC  6000 0952                   BRA     KAIDIS
0203D0  6100 074E                   BSR     B5RN
0203D4  6100 090E           SUP1:   BSR     DRGDIR
0203D8  6100 073C           SUP2:   BSR     B2RN
0203DC  6000 0902                   BRA     KDRDIR
0203E0  6100 073E                   BSR     B5RN
0203E4  6100 0934           SUP3:   BSR     AIPRED
0203E8  6100 072C                   BSR     B2RN
0203EC  6000 0928                   BRA     KAIPRE
0203F0  6100 0724                   BSR     B2RN
0203F4  6000 01C6                   BRA     NCOUNT
0203F8
```

```
0203F8
0203F8  6100 071C              BSR     B2RN
0203FC  6100 08F2              BSR     ARGDIR
020400  6100 071E      SUP4:   BSR     B5RN
020404  6000 08E6              BRA     KARDIR
020408  6100 070C              BSR     B2RN
02040C  6100 08D6              BSR     DRGDIR
020410  6000 FFEE              BRA     SUP4
020414  00000108        TAB6:   DC.L    $00000108
020418  00000148                DC.L    $00000148
02041C  000A0188                DC.L    $000A0188
020420  000A01C8                DC.L    $000A01C8
020424  001A8100                DC.L    $001A8100
020428  002A8108                DC.L    $002A8108
02042C  001AC100                DC.L    $001AC100
020430  002AC108                DC.L    $002AC108
020434  003AC140                DC.L    $003AC140
020438  0042C148                DC.L    $0042C148
02043C  0052C188                DC.L    $0052C188
020440  4D4F5645502E57          DC.B    'MOVEP.W',0
020447  00
020448  4D4F5645502E4C          DC.B    'MOVEP.L',0
02044F  00
020450  4D4F5645502E57          DC.B    'MOVEP.W',0
020457  00
020458  4D4F5645502E4C          DC.B    'MOVEP.L',0
02045F  00
020460  53424344202020          DC.B    'SBCD   ',0
020467  00
020468  53424344202020          DC.B    'SBCD   ',0
02046F  00
020470  41424344202020          DC.B    'ABCD   ',0
020477  00
020478  41424344202020          DC.B    'ABCD   ',0
02047F  00
020480  45584720202020          DC.B    'EXG    ',0
020487  00
020488  45584720202020          DC.B    'EXG    ',0
02048F  00
020490  45584720202020          DC.B    'EXG    ',0
020497  00
020498                  * ================================
020498  6100 0616       GRUP7:  BSR     OPCODE
02049C  6100 0682               BSR     B5RN
0204A0  0243 F0F8               ANDI.W  #$F0F8,D3
0204A4  0C43 50C8               CMPI.W  #$50C8,D3
0204A8  662C                    BNE.S   GRUP8
0204AA  43FA 0BA8               LEA     ABER2(PC),A1
0204AE
```

```
0204AE
0204AE  12FC 0044              MOVE.B  #'D',(A1)+
0204B2  12FC 0042              MOVE.B  #'B',(A1)+
0204B6  3613                   MOVE.W  (A3),D3
0204B8  0243 0F00              ANDI.W  #$0F00,D3
0204BC  EE43                   ASR.W   #7,D3
0204BE  6100 0A96              BSR     CONCOD
0204C2  6100 067C              BSR     NXTWRT
0204C6  6100 081C              BSR     DRGDIR
0204CA  12FC 002C              MOVE.B  #',',(A1)+
0204CE  12FC 0024              MOVE.B  #'$',(A1)+
0204D2  6000 0ACE              BRA     DIST16
0204D6                  * ==============================
0204D6  6100 05D8       GRUP8:  BSR     OPCODE
0204DA  103A 0B3D               MOVE.B  B3(PC),D0
0204DE  0C00 0007               CMPI.B  #7,D0
0204E2  6776                    BEQ.S   GRUP8A
0204E4  3613                    MOVE.W  (A3),D3
0204E6  0243 F138               ANDI.W  #$F138,D3
0204EA  49FA 0032               LEA     TAB8(PC),A4
0204EE  7204                    MOVEQ   #4,D1
0204F0  45FA 000C               LEA     GR8W(PC),A2
0204F4  6100 05F4               BSR     SELECT
0204F8  6660                    BNE.S   GRUP8A
0204FA  6100 0624               BSR     B5RN
0204FE  6100 097E       GR8W:   BSR     BZUWOL
020502  6000 FED0               BRA     SUP1
020506  6100 0976               BSR     BZUWOL
02050A  6000 FED8               BRA     SUP3
02050E  6100 096E               BSR     BZUWOL
020512  6100 07FA               BSR     AIPOST
020516  6100 05FE               BSR     B2RN
02051A  6000 07EE               BRA     KAIPST
02051E  00009100        TAB8:   DC.L    $00009100
020522  00089108                DC.L    $00089108
020526  0010B108                DC.L    $0010B108
02052A  0000D100                DC.L    $0000D100
02052E  0008D108                DC.L    $0008D108
020532  535542582E4220           DC.B    'SUBX.B ',0
020539  00
02053A  535542582E4220           DC.B    'SUBX.B ',0
020541  00
020542  434D504D2E4220           DC.B    'CMPM.B ',0
020549  00
02054A  414444582E4220           DC.B    'ADDX.B ',0
020551  00
020552  414444582E4220           DC.B    'ADDX.B ',0
020559  00
02055A
```

```
02055A
02055A  3613                GRUP8A: MOVE.W   (A3),D3
02055C  6100 05C2                   BSR      B5RN
020560  0243 F8C0                   ANDI.W   #$F8C0,D3
020564  0C43 E0C0                   CMPI.W   #$E0C0,D3
020568  6618                        BNE.S    GRUP8B
02056A  103A 0AAC                   MOVE.B   B2(PC),D0
02056E  4880                        EXT.W    D0
020570  6100 009C                   BSR      MNE8NZ
020574  6172                        BSR.S    GR8RL
020576  12FC 0023                   MOVE.B   #'#',(A1)+
02057A  6100 05AE                   BSR      RNAN
02057E  6000 0760                   BRA      KDRDIR
020582                      * ---------------------------------------
020582  3613                GRUP8B: MOVE.W   (A3),D3
020584  0243 F000                   ANDI.W   #$F000,D3
020588  0C43 E000                   CMPI     #$E000,D3
02058C  6600 008E                   BNE      GRUP9
020590  103A 0A88                   MOVE.B   B4(PC),D0
020594  0240 0003                   ANDI.W   #3,D0
020598  6174                        BSR.S    MNE8NZ
02059A  614C                        BSR.S    GR8RL
02059C  157C 0042 0001              MOVE.B   #'B',01(A2)
0205A2  6100 08DA                   BSR      BZUWOL
0205A6  6100 056E                   BSR      B2RN
0205AA  082B 0005 0001              BTST     #5,01(A3)
0205B0  660A                        BNE.S    NCOUNT
0205B2  12FC 0023                   MOVE.B   #'#',(A1)+
0205B6  6100 057E                   BSR      DATRAN
0205BA  6004                        BRA.S    CDWEIT
0205BC  6100 0726           NCOUNT: BSR      DRGDIR
0205C0  6100 055E           CDWEIT: BSR      B5RN
0205C4  6000 071A                   BRA      KDRDIR
0205C8  41532E2E202020      TAB8A:  DC.B     'AS..    ',0
0205CF  00
0205D0  4C532E2E202020              DC.B     'LS..    ',0
0205D7  00
0205D8  524F582E2E2020              DC.B     'ROX..   ',0
0205DF  00
0205E0  524F2E2E202020              DC.B     'RO..    ',0
0205E7  00
0205E8                      * ---------------------------------------
0205E8  45FA 0A6A           GR8RL:  LEA      ABER2(PC),A2
0205EC  7004                        MOVEQ    #4,D0
0205EE  0C1A 002E           LGR8RL: CMPI.B   #'.',(A2)+
0205F2  57C8 FFFA                   DBEQ     D0,LGR8RL
0205F6  6600 03F0                   BNE      FEHLER
0205FA  157C 004C FFFF              MOVE.B   #'L',-1(A2)
020600
```

```
020600
020600
020600  0813 0000                BTST    #0,(A3)
020604  6606                     BNE.S   NOT8R
020606  062A 0006 FFFF           ADD.B   #6,-1(A2)
02060C  4E75             NOT8R:  RTS
02060E
02060E  E748             MNE8NZ: LSL.W   #3,D0
020610  49FA FFB6               LEA     TAB8A(PC),A4
020614  49F4 0000               LEA     0(A4,D0.W),A4
020618  6000 0570               BRA     MNEGNZ
02061C                   * ================================
02061C  6100 0492        GRUP9:  BSR     OPCODE
020620  6100 04FE               BSR     B5RN
020624  0243 FF00               ANDI.W  #$FF00,D3
020628  49FA 0042               LEA     TAB9(PC),A4
02062C  720C                    MOVEQ   #12,D1
02062E  45FA 000A               LEA     GR9W(PC),A2
020632  6100 04B6               BSR     SELECT
020636  6600 00D0               BNE     GRUP10
02063A  6000 05B4        GR9W:  BRA     IMMBWL
02063E  103A 09D9               MOVE.B  B3(PC),D0
020642  0200 0003               ANDI.B  #3,D0
020646  0C00 0003               CMPI.B  #3,D0
02064A  6700 00BC               BEQ     GRUP10
02064E  6100 082E               BSR     BZUWOL
020652  6000 05F4               BRA     EA01FD
020656  12FC 0024        SPRUMS: MOVE.B  #'$',(A1)+
02065A  103A 09BB               MOVE.B  OC+01(PC),D0
02065E  6704                    BEQ.S   NOTSHT
020660  6000 0926               BRA     SHORT
020664  6100 04DA        NOTSHT: BSR     NXTWRT
020668  6000 0938               BRA     DIST16
02066C  00000000          TAB9: DC.L    $00000000
020670  00000200                DC.L    $00000200
020674  00000400                DC.L    $00000400
020678  00000600                DC.L    $00000600
02067C  00000A00                DC.L    $00000A00
020680  00000C00                DC.L    $00000C00
020684  00044000                DC.L    $00044000
020688  00044200                DC.L    $00044200
02068C  00044400                DC.L    $00044400
020690  00044600                DC.L    $00044600
020694  00044A00                DC.L    $00044A00
020698  001C6000                DC.L    $001C6000
02069C  001C6100                DC.L    $001C6100
0206A0  4F52492E422020          DC.B    'ORI.B  ',0
0206A7  00
0206A8  414E44492E4220          DC.B    'ANDI.B ',0
0206AF  00
0206B0
```

```
0206B0
0206B0  535542492E4220          DC.B    'SUBI.B ',0
0206B7  00
0206B8  414444492E4220          DC.B    'ADDI.B ',0
0206BF  00
0206C0  454F52492E4220          DC.B    'EORI.B ',0
0206C7  00
0206C8  434D50492E4220          DC.B    'CMPI.B ',0
0206CF  00
0206D0  4E4547582E4220          DC.B    'NEGX.B ',0
0206D7  00
0206D8  434C522E422020          DC.B    'CLR.B  ',0
0206DF  00
0206E0  4E45472E422020          DC.B    'NEG.B  ',0
0206E7  00
0206E8  4E4F542E422020          DC.B    'NOT.B  ',0
0206EF  00
0206F0  5453542E422020          DC.B    'TST.B  ',0
0206F7  00
0206F8  425241202020            DC.B    'BRA    ',0
0206FF  00
020700  425352202020            DC.B    'BSR    ',0
020707  00
020708                  * ==============================
020708  6100 03A6       GRUP10: BSR     OPCODE
02070C  6100 0408               BSR     B2RN
020710  0243 F1C0               ANDI.W  #$F1C0,D3
020714  49FA 0050               LEA     TAB10(PC),A4
020718  7211                   MOVEQ   #17,D1
02071A  45FA 000A               LEA     GR10W(PC),A2
02071E  6100 03CA               BSR     SELECT
020722  6600 011A               BNE     GRUP11
020726  6100 05BC       GR10W:  BSR     DRGDIR
02072A  6100 03F4               BSR     B5RN
02072E  6000 0534               BRA     KEA7FD
020732  6100 05B0       SUP5:   BSR     DRGDIR
020736  6100 03E8               BSR     B5RN
02073A  6000 0508               BRA     KEA1FD
02073E  6100 03E0               BSR     B5RN
020742  6100 0538               BSR     EA0FFF
020746  6100 03CE       SUP6:   BSR     B2RN
02074A  6000 05A0               BRA     KARDIR
02074E  6100 03D0       SUP7:   BSR     B5RN
020752  6100 051E               BSR     EA0FFD
020756  6000 FC80               BRA     SUP2
02075A  6100 03C4               BSR     B5RN
02075E  6100 04FE               BSR     EA07F4
020762  6000 FFE2               BRA     SUP6
020766
```

```
020766
020766  00000100          TAB10:  DC.L    $00000100
02076A  000C0140                  DC.L    $000C0140
02076E  000C0180                  DC.L    $000C0180
020772  000C01C0                  DC.L    $000C01C0
020776  00182040                  DC.L    $00182040
02077A  00183040                  DC.L    $00183040
02077E  00284180                  DC.L    $00284180
020782  003441C0                  DC.L    $003441C0
020786  002880C0                  DC.L    $002880C0
02078A  002881C0                  DC.L    $002881C0
02078E  001890C0                  DC.L    $001890C0
020792  001891C0                  DC.L    $001891C0
020796  0018B0C0                  DC.L    $0018B0C0
02079A  0018B1C0                  DC.L    $0018B1C0
02079E  0028C0C0                  DC.L    $0028C0C0
0207A2  0028C1C0                  DC.L    $0028C1C0
0207A6  0018D0C0                  DC.L    $0018D0C0
0207AA  0018D1C0                  DC.L    $0018D1C0
0207AE  42545354202020            DC.B    'BTST   ',0
0207B5  00
0207B6  42434847202020            DC.B    'BCHG   ',0
0207BD  00
0207BE  42434C52202020            DC.B    'BCLR   ',0
0207C5  00
0207C6  42534554202020            DC.B    'BSET   ',0
0207CD  00
0207CE  4D4F5645412E4C            DC.B    'MOVEA.L',0
0207D5  00
0207D6  4D4F5645412E57            DC.B    'MOVEA.W',0
0207DD  00
0207DE  43484B20202020            DC.B    'CHK    ',0
0207E5  00
0207E6  4C454120202020            DC.B    'LEA    ',0
0207ED  00
0207EE  44495655202020            DC.B    'DIVU   ',0
0207F5  00
0207F6  44495653202020            DC.B    'DIVS   ',0
0207FD  00
0207FE  535542412E5720            DC.B    'SUBA.W ',0
020805  00
020806  535542412E4C20            DC.B    'SUBA.L ',0
02080D  00
02080E  434D50412E5720            DC.B    'CMPA.W ',0
020815  00
020816  434D50412E4C20            DC.B    'CMPA.L ',0
02081D  00
02081E  4D554C55202020            DC.B    'MULU   ',0
020825  00
020826
```

```
020826
020826  4D554C53202020          DC.B    'MULS   ',0
02082D  00
02082E  414444412E5720          DC.B    'ADDA.W ',0
020835  00
020836  414444412E4C20          DC.B    'ADDA.L ',0
02083D  00
02083E                  * ==============================
02083E  6100 0270       GRUP11: BSR     OPCODE
020842  0243 F0C0               ANDI.W  #$F0C0,D3
020846  0C43 50C0               CMPI.W  #$50C0,D3
02084A  6618                    BNE.S   GRUP12
02084C  43FA 0806               LEA     ABER2(PC),A1
020850  12FC 0053               MOVE.B  #'S',(A1)+
020854  6100 06F6               BSR     SETCOD
020858  6100 02C6               BSR     B5RN
02085C  363A 07B8               MOVE.W  0C(PC),D3
020860  6000 03E6               BRA     EA01FD
020864                  * ==============================
020864  6100 024A       GRUP12: BSR     OPCODE
020868  0243 F100               ANDI.W  #$F100,D3
02086C  49FA 006C               LEA     TAB12(PC),A4
020870  720C                    MOVEQ   #12,D1
020872  45FA 000A               LEA     GR12W(PC),A2
020876  6100 0272               BSR     SELECT
02087A  6600 00FA               BNE     GRUP13
02087E  12FC 0023       GR12W:  MOVE.B  #'#',(A1)+
020882  12FC 0024               MOVE.B  #'$',(A1)+
020886  6100 02E2               BSR     WBYTAN
02088A  6000 FB4C               BRA     SUP2
02088E  6100 05EE               BSR     BZUWOL
020892  12FC 0023               MOVE.B  #'#',(A1)+
020896  6100 029C               BSR     B2DATA
02089A  6100 0284               BSR     B5RN
02089E  6000 03AE               BRA     KEA1FF
0208A2  6100 05DA               BSR     BZUWOL
0208A6  6000 FEA6               BRA     SUP7
0208AA  6100 05D2               BSR     BZUWOL
0208AE  6100 0266               BSR     B2RN
0208B2  6100 0430               BSR     DRGDIR
0208B6  6100 0268               BSR     B5RN
0208BA  6000 037E               BRA     KEA1FC
0208BE  6100 05BE               BSR     BZUWOL
0208C2  6100 025C               BSR     B5RN
0208C6  6100 03B4               BSR     EA0FFF
0208CA  6000 FB0C               BRA     SUP2
0208CE  6100 05AE               BSR     BZUWOL
0208D2  6100 0242               BSR     B2RN
0208D6  6000 FE5A               BRA     SUP5
0208DA
```

```
0208DA
0208DA  00007000          TAB12:  DC.L    $00007000
0208DE  00105000                  DC.L    $00105000
0208E2  00105100                  DC.L    $00105100
0208E6  00248000                  DC.L    $00248000
0208EA  002C8100                  DC.L    $002C8100
0208EE  00409000                  DC.L    $00409000
0208F2  002C9100                  DC.L    $002C9100
0208F6  0040B000                  DC.L    $0040B000
0208FA  0050B100                  DC.L    $0050B100
0208FE  0024C000                  DC.L    $0024C000
020902  002CC100                  DC.L    $002CC100
020906  0040D000                  DC.L    $0040D000
02090A  002CD100                  DC.L    $002CD100
02090E  4D4F5645512020            DC.B    'MOVEQ  ',0
020915  00
020916  414444512E4220             DC.B    'ADDQ.B ',0
02091D  00
02091E  535542512E4220            DC.B    'SUBQ.B ',0
020925  00
020926  4F522E42202020            DC.B    'OR.B   ',0
02092D  00
02092E  4F522E42202020            DC.B    'OR.B   ',0
020935  00
020936  5355422E422020            DC.B    'SUB.B  ',0
02093D  00
02093E  5355422E422020            DC.B    'SUB.B  ',0
020945  00
020946  434D502E422020            DC.B    'CMP.B  ',0
02094D  00
02094E  454F522E422020            DC.B    'EOR.B  ',0
020955  00
020956  414E442E422020            DC.B    'AND.B  ',0
02095D  00
02095E  414E442E422020            DC.B    'AND.B  ',0
020965  00
020966  4144442E422020            DC.B    'ADD.B  ',0
02096D  00
02096E  4144442E422020            DC.B    'ADD.B  ',0
020975  00
020976                    * ============================
020976  6100 0138         GRUP13: BSR     OPCODE
02097A  6100 01A4                 BSR     B5RN
02097E  0243 F000                 ANDI.W  #$F000,D3
020982  49FA 002A                 LEA     TAB13(PC),A4
020986  323C 0002                 MOVE    #2,D1
02098A  45FA 0008                 LEA     GR13W(PC),A2
02098E
```

```
02098E
02098E  6100 015A                BSR      SELECT
020992  663E                     BNE.S    GRUP14
020994  6100 02E6       GR13W:   BSR      EA0FFF
020998  45FA 067E                LEA      B2(PC),A2
02099C  1552 0004                MOVE.B   (A2),04(A2)
0209A0  156A 0001 0002           MOVE.B   01(A2),02(A2)
0209A6  1552 0003                MOVE.B   (A2),03(A2)
0209AA  6000 0298                BRA      KEA1FD
0209AE  00001000        TAB13:   DC.L     $00001000
0209B2  00002000                 DC.L     $00002000
0209B6  00003000                 DC.L     $00003000
0209BA  4D4F56452E4220           DC.B     'MOVE.B ',0
0209C1  00
0209C2  4D4F56452E4C20           DC.B     'MOVE.L ',0
0209C9  00
0209CA  4D4F56452E5720           DC.B     'MOVE.W ',0
0209D1  00
0209D2                  * ==================================
0209D2  0C43 6000       GRUP14:  CMPI.W   #$6000,D3
0209D6  6610                     BNE.S    FEHLER
0209D8  43FA 067A                LEA      ABER2(PC),A1
0209DC  12FC 0042                MOVE.B   #'B',(A1)+
0209E0  6100 056A                BSR      SETCOD
0209E4  6000 FC70                BRA      SPRUMS
0209E8                  * ==================================
0209E8  49FA 0020       FEHLER:  LEA      TFEHL(PC),A4
0209EC  43FA 066E                LEA      ABER3(PC),A1
0209F0  6100 018E                BSR      TEXTAN
0209F4  200F                     MOVE.L   A7,D0
0209F6  C348                     EXG      A1,A0
0209F8  4EB9 0000164A            JSR      @PRINT8X
0209FE  C149                     EXG      A0,A1
020A00  45FA F64A                LEA      LAUSGB(PC),A2
020A04  2E7A 0604                MOVEA.L  DEPA7(PC),A7
020A08  4ED2                     JMP      (A2)
020A0A  3C3C204645484C  TFEHL:   DC.B     '<< FEHLER << ',0
020A11  4552203C3C2000
020A18
020A18  4E75            FINIS:   RTS
020A1A                  **********************************
020A1A  41FA 000C       DEFJOB:  LEA      TJOB(PC),A0
020A1E  612E                     BSR.S    EINLES
020A20  41FA 05FB                LEA      JOB(PC),A0
020A24  1080                     MOVE.B   D0,(A0)
020A26  4E75                     RTS
020A28  4A4F4200        TJOB:    DC.B     'JOB',0
020A2C
```

```
020A2C
020A2C  41FA 0018          GRENZN: LEA     TVON(PC),A0
020A30  611C                       BSR.S   EINLES
020A32  41FA 05DA                  LEA     VON(PC),A0
020A36  2080                       MOVE.L  D0,(A0)
020A38  41FA 0010                  LEA     TBIS(PC),A0
020A3C  6110                       BSR.S   EINLES
020A3E  41FA 05D2                  LEA     BIS(PC),A0
020A42  2080                       MOVE.L  D0,(A0)
020A44  4E75                       RTS
020A46  564F4E00           TVON:   DC.B    'VON',0
020A4A  42495300           TBIS:   DC.B    'BIS',0
020A4E
020A4E  4EB9 00003FEC      EINLES: JSR     @CLRSCREEN
020A54  323C 0001                  MOVE.W  #1,D1
020A58  343C 0001                  MOVE.W  #1,D2
020A5C  4EB9 000016BE              JSR     @SETCURXY
020A62  6100 017A                  BSR     LAUSG1
020A66  7011                       MOVEQ   #$11,D0
020A68  323C 0028                  MOVE.W  #40,D1
020A6C  343C 00EC                  MOVE.W  #236,D2
020A70  363C 0007                  MOVE.W  #7,D3
020A74  41FA 05B8                  LEA     ABER1(PC),A0
020A78  4EB9 00001486              JSR     @READ
020A7E  41FA 05AE                  LEA     ABER1(PC),A0
020A82  4EF9 0000172C              JMP     @WERT
020A88
020A88  41FA 05A4          ADRES:  LEA     ABER1(PC),A0
020A8C  7610                       MOVEQ   #16,D3
020A8E  20FC 20202020      LOPAD1: MOVE.L  #$20202020,(A0)+
020A94  51CB FFF8                  DBRA    D3,LOPAD1
020A98  20BC 00000000              MOVE.L  #$0,(A0)
020A9E  41FA 058E                  LEA     ABER1(PC),A0
020AA2  200B                       MOVE.L  A3,D0
020AA4  4EB9 0000164A              JSR     @PRINT8X
020AAA  30FC 2020                  MOVE.W  #$2020,(A0)+
020AAE  4E75                       RTS
020AB0
```

```
020AB0
020AB0  3613              OPCODE: MOVE.W   (A3),D3
020AB2  45FA 0562                 LEA      0C(PC),A2
020AB6  34C3                      MOVE.W   D3,(A2)+
020AB8  3003                      MOVE.W   D3,D0
020ABA  0280 00000E00             ANDI.L   #$0E00,D0
020AC0  80FC 0200                 DIVU     #$200,D0
020AC4  14C0                      MOVE.B   D0,(A2)+
020AC6  3003                      MOVE.W   D3,D0
020AC8  0280 000001C0             ANDI.L   #$01C0,D0
020ACE  80FC 0040                 DIVU     #$40,D0
020AD2  14C0                      MOVE.B   D0,(A2)+
020AD4  1003                      MOVE.B   D3,D0
020AD6  0280 00000038             ANDI.L   #$0038,D0
020ADC  E608                      LSR.B    #3,D0
020ADE  14C0                      MOVE.B   D0,(A2)+
020AE0  1003                      MOVE.B   D3,D0
020AE2  0200 0007                 ANDI.B   #7,D0
020AE6  14C0                      MOVE.B   D0,(A2)+
020AE8  4E75                      RTS
020AEA
020AEA  3801              SELECT: MOVE.W   D1,D4
020AEC  2A1C              SELLOP: MOVE.L   (A4)+,D5
020AEE  B645                      CMP.W    D5,D3
020AF0  57CC FFFA                 DBEQ     D4,SELLOP
020AF4  661E                      BNE.S    SELLEX
020AF6  4845                      SWAP     D5
020AF8  3001                      MOVE.W   D1,D0
020AFA  D041                      ADD.W    D1,D0
020AFC  9044                      SUB.W    D4,D0
020AFE  E548                      LSL.W    #2,D0
020B00  49F4 0000                 LEA      00(A4,D0.W),A4
020B04  6100 0084                 BSR      MNEGNZ
020B08  49F2 5000                 LEA      00(A2,D5.W),A4
020B0C  DFFC 00000004             ADDA.L   #4,A7
020B12  4ED4                      JMP      (A4)
020B14  4E75              SELLEX: RTS
020B16
020B16  45FA 0500         B2RN:   LEA      B2(PC),A2
020B1A  1552 0004                 MOVE.B   (A2),04(A2)
020B1E  4E75                      RTS
020B20
020B20  45FA 04F9         B5RN:   LEA      B5(PC),A2
020B24  1552 0001                 MOVE.B   (A2),01(A2)
020B28  4E75                      RTS
020B2A
```

Zum paarweisen Erkennen genügt die 15-bit-Maske FFBF. Zur Erkennung des Ziels muß anschließend Bit6 abgefragt werden. Das Ergebnis (in Z) wird mit in die Routine CKSRCR, die Constante und Komma und SR oder CCR anfügt, übernommen. War das Bit gesetzt (also Not EQu), dann geht es weiter bei CONKSR (Wort-Constante, Komma und SR); anderenfalls bei CNKCCR (Byte-CoNstante, Komma CCR). Alle diese Mikroabschnitte führen letztlich mit einem RTS zurück ins Hauptprogramm, wo – wie bereits besprochen – auf DECODE AUSGEB folgt.

29.8 Die TRAP-Befehle (GRUP3: 20124)

Die Codierung der sechzehn TRAP-Befehle ist 4E4N, worin N, das LSN, die Trap-Nummer darstellt. Hier sei lediglich mitgeteilt, daß NIBMGS bedeutet «NIBble Mit Gitter und Schlange».

29.9 Die Befehle der Gruppe 4 (20148)

Die Opcodes dieser Gruppe haben zweierlei gemeinsam: Fünfzehn Bits sind festgelegt. Das erste Nibble ist 4; hiernach wird zuerst gefragt. Nach dem Auffinden der Mnemonics geht es so weiter:

SWAP: Der Zieloperand ist «DatenReGister DIRekt» adressiert. Die Ausformulierung übernimmt die Subroutine DRGDIR; das scheinbar am Ende der Subroutine fehlende RTS beschließt RNAN (RegisterNummer ANfügen). Dieser Mikroabschnitt holt die Registernummer aus dem RN-Feld, wo sie natürlich vorher bereitgestellt werden mußte, was hier B5RN (b5 wird zur Registernummer) bewirkte. NIBAS bereitet dieses NIBble zum ASCII-Code auf. Der Einsprung in diese Aufbereitungssequenz ist in GR4W+00 und gilt für SWAP, EXT.W und EXT.L (siehe TAB4).

LINK: Die Syntax lautet LINK An,#Const. Dementsprechend ist die Aufbereitungssequenz so aufgebaut: ARGDIR (AdreßReGister DIRekt), KCNBWL (Komma, CoNstante vom Format Byte, Wort oder Langwort (wie bereits besprochen)). Einsprung bei GR4W+04.

KUSPAN bedeutet Komma und USP ANfügen,

USPRAN rückt USP RAN und

KARDIR Komma und AdreßRegister DIRekt.

```
020B2A
020B2A  103A 04F0       RNAN:   MOVE.B  RN(PC),D0
020B2E  616A            RNAN2:  BSR.S   NIBAS
020B30  12C0                    MOVE.B  D0,(A1)+
020B32  4E75                    RTS
020B34  61E0            B2DATA: BSR.S   B2RN
020B36  103A 04E4       DATRAN: MOVE.B  RN(PC),D0
020B3A  66F2                    BNE.S   RNAN2
020B3C  5000                    ADDQ.B  #8,D0
020B3E  60EE                    BRA.S   RNAN2
020B40
020B40  47EB 0002       NXTWRT: LEA     02(A3),A3
020B44  4DFA 04D8               LEA     NW(PC),A6
020B48  3013                    MOVE.W  (A3),D0
020B4A  3C80                    MOVE.W  D0,(A6)
020B4C  10FC 0020               MOVE.B  #' ',(A0)+
020B50  3013            WORTAN: MOVE.W  (A3),D0
020B52  4EB9 00001660           JSR     @PRINT4X
020B58  10BC 0020               MOVE.B  #$20,(A0)
020B5C  4E75                    RTS
020B5E  12FC 0024       WRT2MS: MOVE.B  #'$',(A1)+
020B62  C149            WRT2AN: EXG     A0,A1
020B64  61EA                    BSR.S   WORTAN
020B66  C348                    EXG     A1,A0
020B68  4E75                    RTS
020B6A
020B6A  C149            WBYTAN: EXG     A0,A1
020B6C  3013                    MOVE.W  (A3),D0
020B6E  4EB9 00001668           JSR     @PRINT2X
020B74  10BC 0020               MOVE.B  #$20,(A0)
020B78  C348                    EXG     A1,A0
020B7A  4E75                    RTS
020B7C
020B7C  43FA 04D6       MNERAN: LEA     ABER2(PC),A1
020B80  12DC            TEXTAN: MOVE.B  (A4)+,(A1)+
020B82  66FC                    BNE.S   TEXTAN
020B84  133C 0020               MOVE.B  #$20,-(A1)
020B88  4E75                    RTS
020B8A  61F0            MNEGNZ: BSR.S   MNERAN
020B8C  43FA 04CE               LEA     ABER3(PC),A1
020B90  4E75                    RTS
020B92
020B92  12FC 0023       NIBMGS: MOVE.B  #'#',(A1)+
020B96  12FC 0024               MOVE.B  #'$',(A1)+
020B9A  0240 000F       NIBAS:  ANDI.W  #$000F,D0
020B9E  0600 0030               ADDI.B  #$30,D0
020BA2  0C00 003A               CMPI.B  #$3A,D0
020BA6  6502                    BCS.S   NIBEX
020BA8  5E40                    ADDQ    #7,D0
020BAA  4E75            NIBEX:  RTS
020BAC
```

29.10 Die Superroutine EAXXEX (20C80)

Der Name ist «geschichtlich» bedingt; das Programm wurde zunächst auf dem MZ-700 in Sharp-BASIC entworfen, dann für den QL in Sinclair-SuperBASIC übertragen und schließlich auf dem NDR in Maschinensprache realisiert. EAXXEX vereinheitlicht die Ausführung (EXecution) der Ausformulierung der zugelassenen Effektiven Adressen.

Die Tabelle TEAXX enthält alle zwölf regulären Adressierungsarten. Die ersten sieben werden bei vielen Befehlen durch b4 = 0 ... 6 definiert; b5 ist dann die Registernummer. Für die weiteren fünf Adressierungsarten ist b4 = 7; hier definiert die Summe b4 + b5 = 7 die Adressierungsart «absolut.W», die Summe 8 «absolut.L», die Summe 9 «d(PC)» usw. Für den Befehl CHK <ea>,Dn sind alle Adressierungsarten bis auf An zugelassen. Dies symbolisiert in kürzester Form die Leitzahl 1111 1111 1101 = 0FFD. Diesen Fall wickelt der Programmteil EA0FFD ab oder – wenn noch ein Komma davor gesetzt werden soll – die Routine KEAFFD. Sie werden diesen Ansatz sicherlich in der Gruppe 10, z.B. für CHK, finden.

Wenn – in diesem Beispiel – nun aber b4 = 1 die nicht zugelassene Adressierungsart Adreßregister direkt anfordert, «erkennt» EAXXEX die Null in Bit1 der Leitzahl und löst eine Fehlermeldung aus.

Der Ablauf von EAXXEX ist nun nicht mehr schwer nachzuvollziehen.

29.11 Weitere Routinen

Der Befehl ADDQ kann 3-bit-Daten zu <ea> addieren. Da die Addition von 0 wenig sinnvoll wäre, codiert die 0 den Summanden 8. Diese 0-zu-8-Aufwertung nebst ASCII-Codierung bewirkt die Routine DATRAN (DATen RANrücken).

WRT2MS fügt das zweite Wort Mit Schlange an, z.B. eine absolute 16-bit-Adresse.

WBYTAN fügt das untere BYTe eines Worts als String AN. MNERAN fügt einen erkannten MNEmonic als String RAN. Anfangs muß A4 auf den Stringanfang zeigen und A1 auf ABER2. Ohne das letztere bleibt TEXTAN übrig. MNEGNZ hat mit dem Anfügen des MNEmonics schon die GaNZe Arbeit in ABER2 getan und richtet danach A1 auf ABER3.

PRELDI bearbeitet die Adressierungsart «Programmzähler RELativ mit AdreßDIstanz».

```
020BAC
020BAC  103A 046F        AUSGEB: MOVE.B  JOB(PC),D0
020BB0  0200 0003                ANDI.B  #3,D0
020BB4  0C00 0001                CMPI.B  #1,D0
020BB8  6720                     BEQ.S   DISAUS
020BBA  0C00 0002                CMPI.B  #2,D0
020BBE  670C                     BEQ.S   PRIAUS
020BC0  0C00 0003                CMPI.B  #3,D0
020BC4  6604                     BNE.S   AUSGEX
020BC6  6112                     BSR.S   DISAUS
020BC8  6002                     BRA.S   PRIAUS
020BCA  4E75             AUSGEX: RTS
020BCC  4EB9 000031A8    PRIAUS: JSR     @LST
020BD2  6106                     BSR.S   DISAUS
020BD4  4EF9 000031A0            JMP     @CRT
020BDA  41FA 0454        DISAUS: LEA     ABER1+02(PC),A0
020BDE  1018             LAUSG1: MOVE.B  (A0)+,D0
020BE0  6708                     BEQ.S   AUSEX
020BE2  4EB9 00000AE0            JSR     @CO2
020BE8  60F4                     BRA.S   LAUSG1
020BEA  4EF9 00001698    AUSEX:  JMP     @CRLF
020BF0
020BF0  6100 028C        IMMBWL: BSR     BZUWOL
020BF4  49FA 0014                LEA     TIMBWL(PC),A4
020BF8  103A 041F                MOVE.B  B3(PC),D0
020BFC  0240 0003                ANDI.W  #3,D0
020C00  E540                     ASL.W   #2,D0
020C02  2874 0000                MOVEA.L 00(A4,D0.W),A4
020C06  4E94                     JSR     (A4)
020C08  603A                     BRA.S   KEA1FD
020C0A  00020DFA         TIMBWL: DC.L    BYTMGS
020C0E  00020E2E                 DC.L    CONBWL
020C12  00020E1A                 DC.L    LAWMGS
020C16  000209E8                 DC.L    FEHLER
020C1A
020C1A  660C             CKSRCR: BNE.S   CONKSR
020C1C  6100 01DC        CNKCCR: BSR     BYTMGS
020C20  12FC 002C                MOVE.B  #',',(A1)+
020C24  6000 024A                BRA     CCRRAN
020C28
020C28  6100 0204        CONKSR: BSR     CONBWL

020C2C  6000 0230                BRA     KSRGAN
020C30
020C30  12FC 002C        KEA1F4: MOVE.B  #',',(A1)+
020C34  363C 01F4        EA01F4: MOVE.W  #$01F4,D3
020C38  6046                     BRA.S   EAXXEX
020C3A
```

Interessierte Leser werden sicherlich viele andere Routinen deuten können, denn es wurde versucht, sie ziemlich konsequent nach ihrer Funktion zu benennen.

29.12 Die Decodierung der Registerliste (20EAE)

Auf den Opcode von MOVEM (Gruppe 5a) folgt die 16-bit-Codierung der Registerliste. Sie sieht fast immer so aus:

```
15  14  13  ...  08  07  ...  02  01  00  = Bit-Nr.
A7  A6  A5  ...  A0  D7  ...  D2  D1  D0
```

Die Codierung 00FF besagt also, daß alle Datenregister abzulegen oder zu holen sind.

Nur bei der Adressierungsart Adreßregister indirekt mit Predekrement wird in umgekehrter Reihenfolge codiert:

```
15  14  13  ...  08  07  ...  02  01  00  = Bit-Nr.
D0  D1  D2  ...  D7  A0  ...  A5  A6  A7
```

Man kann nach einem gemeinsamen Algorithmus decodieren, wenn man vor dem Aufbau der lesbaren Registerliste die eine der beiden Codierungen spiegelt, indem man die Bits 0 und 15, 1 und 14 usw. austauscht, was die Routine INVLIS (INVertiere registerLISte) bewirkt. Wir behalten nur im Falle −(An) die ursprüngliche Codierung bei und spiegeln in allen anderen. Nachdem in Gruppe 5A der Opcode erkannt, der Mnemonik nach ABER2 übertragen und erkannt und berücksichtigt wurde, ob im .W- oder im .L-Format transferiert werden soll, wird das nächste Datenwort, die Codierung der Registerliste, geholt und vom Feld NW ins Feld CODLIS (CODeLISte) kopiert. Ist Bit2 des Opcodes gesetzt, wird vom Speicher in die Register übertragen, und es geht weiter bei MEMREG, anderenfalls bei REGMEM. In diesem Falle muß die Adressierungsart «Adreßregister indirekt mit Predekrement» erkannt werden. Sie liegt vor, wenn b4 = 4 ist, und dann geht es weiter mit REGLIS zur Decodierung der Registerliste. In allen anderen Fällen muß vorher gespiegelt werden.

Nehmen wir nun an,

1110 1011 0000 0000 sei zu D0-D2/D4/D6-D7

zu decodieren. A4 weist auf CODLIS. REGLIS holt zunächst die Listencodierung nach D3 und lädt D5 mit ' A D' = $20412044, so daß sich hieraus ganz

leicht die benötigten As und Ds in die Registerliste übertragen lassen. D3 wird nun (unnötigerweise) auf Langwort erweitert und um acht Binärstellen nach rechts rotiert: D3 = 0000 0000 1111 1111 1111 1111 1110 1011. Das LSB von D3 enthält jetzt also die Codierung der Datenregister, die von ANALY analysiert wird: Zunächst wird ein zwölf Byte breiter Puffer gelöscht. Nach Passieren von LANA1 ist D4 = 0. Das LSB von D3 wird getestet. Da das ganze Byte nicht gleich 0 ist, gibt es etwas zu decodieren. Das Byte wird um eine Binärstelle nach links geshiftet. Wäre nun D0 nicht in der Liste verzeichnet, wäre C-Flag gleich 0, und es ginge zurück nach LANA1, wo der Bitzähler D4 um 1 erhöht würde. Neuer Durchlauf.

```
020C3A  12FC 002C       KEA1FC: MOVE.B  #',',(A1)+
020C3E  363C 01FC       EA01FC: MOVE.W  #$01FC,D3
020C42  603C                    BRA.S   EAXXEX
020C44
020C44  12FC 002C       KEA1FD: MOVE.B  #',',(A1)+
020C48  363C 01FD       EA01FD: MOVE.W  #$01FD,D3
020C4C  6032                    BRA.S   EAXXEX
020C4E
020C4E  12FC 002C       KEA1FF: MOVE.B  #',',(A1)+
020C52  363C 01FF       EA01FF: MOVE.W  #$01FF,D3
020C56  6028                    BRA.S   EAXXEX
020C58
020C58  363C 07EC       EA07EC: MOVE.W  #$07EC,D3
020C5C  6022                    BRA.S   EAXXEX
020C5E
020C5E  363C 07F4       EA07F4: MOVE.W  #$07F4,D3
020C62  601C                    BRA.S   EAXXEX
020C64
020C64  12FC 002C       KEA7FD: MOVE.B  #',',(A1)+
020C68  363C 07FD       EA07FD: MOVE.W  #$07FD,D3
020C6C  6012                    BRA.S   EAXXEX
020C6E
020C6E  12FC 002C       KEAFFD: MOVE.B  #',',(A1)+
020C72  363C 0FFD       EA0FFD: MOVE.W  #$0FFD,D3
020C76  6008                    BRA.S   EAXXEX
020C78
```

Symboltabelle:
Damit man in längeren Listings schnell eine bestimmte Stelle auffinden kann, liefern gute Assembler eine Aufstellung der verwendeten Labels und der ihnen zugeordneten Adressen (s. Seite 250).

A.29.1: Was bedeuten die Befehle in der zweiten bis sechsten Zeile des Disassemblerprogramms?

In unserem Falle wurde aber bei D4 = 0 schon C-Flag gesetzt: Die erste 1 wurde gefunden, und es geht weiter bei GEFUN1.

GEFUN1: A2 wird auf das nächste (anfangs also das erste) Feld des Puffers gesetzt, dessen Inhalt um 1 erhöht (anfangs also auf 1 gesetzt) wird. D3.B = 1101 0110, also ≠ 0. Deshalb also weiter mit erneutem ASL.B #1,D3, wonach D3.B = 1010 1100 und C = 1. Deshalb zurück nach LGEF11 und Wiederholung, bis schließlich das erste Feld des Puffers 3 enthält und D3.B = 0101 1000 und C = 0. Jetzt geht es weiter bei GEFUN2.

GEFUN2: Hier werden nun in gleicher Weise die sich an die Einsen anschließenden Nullen in der codierten Registerliste gezählt; es ist nur eine. Dann wieder zurück nach GEFUN1.

Dieses Wechselspiel wird fortgesetzt, bis D3.B leergeshiftet wurde. Der Puffer enthält nun 31112, einen Zwischencode, der in unserem Beispiel besagt: Auf 3 Einsen folgen 1 Null, 1 Eins, 1 Null und 2 Einsen. Dieser Umweg erschien angebracht, um z.B. D3/D4/D5 leichter zu D3–D5 komprimieren zu können.

Nach dieser Aufbereitung geht es zum AUFBAU der lesbaren Registerliste: Mit Postinkrement wird die erste Ziffer des Zwischencodes (die 3) nach D0 geholt. Solange das keine 0 ist, ist dessen Ende noch nicht erreicht. Aus D5 wird ein «D» nach ABER3 übertragen, und aus D4 die von vorn gezählte Ordnungszahl der ersten erreichten 1 in der codierten Registerliste. Diese muß natürlich durch Addition von $30 noch zum entsprechenden ASCII erhöht werden. Enthält D0 nun eine 1, dann geht es weiter bei NUR1, und es müßte «/» angefügt werden. In unserem Falle folgen aber mehrere (nämlich drei) Datenregister aufeinander; dementsprechend folgen das Zeichen «–» und ein «D». Nach der Addition der Anzahl der Einsen aus D0 (3) zur fortlaufenden Zählung der Einsen in D4 wird die Summe 3 nach ABER übertragen. Da sie aber um 1 zu groß ist (D0–D2 soll gebildet werden), wird nur $2F zum richtigen ASCII aufaddiert und «/» angefügt.

Es erscheint wenig anregend, jedes Detail in epischer Breite auszuführen. Interessierte Leser werden nun die Feinheiten selber herausfinden und mit dem Programm experimentieren können, z.B. durch Einbau eines RTS oder eines STOP an interessanten Stellen. Der NDR-Computer bietet hier ja eine Vielzahl von Möglichkeiten.

Die Skizzierung des weiteren Gedankengangs sei mit dem Hinweis beendet, daß die Adreßregisterliste nach genau gleichem Verfahren erzeugt wird, natürlich nach entsprechendem Zurückrotieren von D3.L und «von-D-zu-A-Swappen» von D5.

```
020C78
020C78  12FC 002C       KEAFFF: MOVE.B  #',',(A1)+
020C7C  363C 0FFF       EA0FFF: MOVE.W  #$0FFF,D3
020C80  143A 0398       EAXXEX: MOVE.B  B4(PC),D2
020C84  0C02 0007               CMPI.B  #7,D2
020C88  6608                    BNE.S   NOTADX
020C8A  143A 038F               MOVE.B  B5(PC),D2
020C8E  0602 0007               ADDI.B  #7,D2
020C92  720B            NOTADX: MOVEQ   #11,D1
020C94  B401            SUMLOP: CMP.B   D1,D2
020C96  6708                    BEQ.S   MODUS
020C98  51C9 FFFA               DBRA    D1,SUMLOP
020C9C  6000 FD4A               BRA     FEHLER
020CA0  4882            MODUS:  EXT.W   D2
020CA2  0503                    BTST    D2,D3
020CA4  6700 FD42               BEQ     FEHLER
020CA8  E54A                    LSL.W   #2,D2
020CAA  247B 2004               MOVEA.L TEAXX(PC,D2.W),A2
020CAE  4ED2                    JMP     (A2)
020CB0  00020CE4        TEAXX:  DC.L    DRGDIR  * DN
020CB4  00020CF0                DC.L    ARGDIR  * AN
020CB8  00020CF8                DC.L    ARINDI  * (AN)
020CBC  00020D0E                DC.L    AIPOST  * (AN)+
020CC0  00020D1A                DC.L    AIPRED  * -(AN)
020CC4  00020D24                DC.L    AIDIST  * D(AN)
020CC8  00020D28                DC.L    AIDIND  * D(AN,RX.X)
020CCC  00020D7A                DC.L    ABSWRT  * $XXXX
020CD0  00020E1E                DC.L    ABSLAN  * $XXXXXXXX
020CD4  00020D86                DC.L    PRELDI  * D(PC)
020CD8  00020DB4                DC.L    PRDIND  * D(PC,RX.X)
020CDC  00020E2E                DC.L    CONBWL  * #.B.W.L
020CE0
020CE0  12FC 002C       KDRDIR: MOVE.B  #',',(A1)+
020CE4  12FC 0044       DRGDIR: MOVE.B  #'D',(A1)+
020CE8  6000 FE40               BRA     RNAN
020CEC
020CEC  12FC 002C       KARDIR: MOVE.B  #',',(A1)+
020CF0  12FC 0041       ARGDIR: MOVE.B  #'A',(A1)+
020CF4  6000 FE34               BRA     RNAN
020CF8
020CF8  12FC 0028       ARINDI: MOVE.B  #'(',(A1)+
020CFC  12FC 0041               MOVE.B  #'A',(A1)+
020D00  6100 FE28               BSR     RNAN
020D04  12FC 0029               MOVE.B  #')',(A1)+
020D08  4E75                    RTS
020D0A
020D0A  12FC 002C       KAIPST: MOVE.B  #',',(A1)+
020D0E  61E8            AIPOST: BSR.S   ARINDI
020D10  12FC 002B               MOVE.B  #'+',(A1)+
020D14  4E75                    RTS
020D16
```

```
020D16
020D16  12FC 002C       KAIPRE: MOVE.B  #',',(A1)+
020D1A  12FC 002D       AIPRED: MOVE.B  #'-',(A1)+
020D1E  60D8                    BRA.S   ARINDI
020D20
020D20  12FC 002C       KAIDIS: MOVE.B  #',',(A1)+
020D24  6154            AIDIST: BSR.S   ABSWRT
020D26  60D0                    BRA.S   ARINDI
020D28
020D28  6100 FE16       AIDIND: BSR     NXTWRT
020D2C  12FC 0024               MOVE.B  #'$',(A1)+
020D30  6100 FE38               BSR     WBYTAN
020D34  12FC 0028               MOVE.B  #'(',(A1)+
020D38  12FC 0041               MOVE.B  #'A',(A1)+
020D3C  6100 FDEC               BSR     RNAN
020D40  12FC 002C               MOVE.B  #',',(A1)+
020D44  12FC 0041               MOVE.B  #'A',(A1)+
020D48  4A13                    TST.B   (A3)
020D4A  6B06                    BMI.S   NICHTD
020D4C  0629 0003 FFFF          ADD.B   #3,-1(A1)
020D52  1013            NICHTD: MOVE.B  (A3),D0
020D54  0200 0070               ANDI.B  #$70,D0
020D58  E800                    ASR.B   #4,D0
020D5A  0600 0030               ADD.B   #$30,D0
020D5E  12C0                    MOVE.B  D0,(A1)+
020D60  12FC 002E               MOVE.B  #'.',(A1)+
020D64  12FC 004C               MOVE.B  #'L',(A1)+
020D68  0813 0003               BTST    #3,(A3)
020D6C  6606                    BNE.S   NICHTW
020D6E  0629 000B FFFF          ADD.B   #11,-1(A1)
020D74  12FC 0029       NICHTW: MOVE.B  #')',(A1)+
020D78  4E75                    RTS
020D7A
020D7A  6100 FDC4       ABSWRT: BSR     NXTWRT
020D7E  12FC 0024               MOVE.B  #'$',(A1)+
020D82  6000 FDDE               BRA     WRT2AN
020D86
020D86  6100 FDB8       PRELDI: BSR     NXTWRT
020D8A  12FC 0024               MOVE.B  #'$',(A1)+
020D8E  6100 FDD2               BSR     WRT2AN
020D92  12FC 0028               MOVE.B  #'(',(A1)+
020D96  12FC 0050               MOVE.B  #'P',(A1)+
020D9A  12FC 0043               MOVE.B  #'C',(A1)+
020D9E  12FC 0029               MOVE.B  #')',(A1)+
020DA2  2F09                    MOVE.L  A1,-(A7)
020DA4  43FA 02CA               LEA     ABER3+20(PC),A1
020DA8  12FC 003D               MOVE.B  #'=',(A1)+
020DAC  6100 01F4               BSR     DIST16
020DB0  225F                    MOVEA.L (A7)+,A1
020DB2  4E75                    RTS
020DB4
```

```
020DB4
020DB4  6100 FF72       PRDIND: BSR     AIDIND
020DB8  7210                    MOVEQ   #16,D1
020DBA  45FA 029C               LEA     ABER2+4(PC),A2
020DBE  0C1A 0028       LPRDIN: CMPI.B  #'(',(A2)+
020DC2  57C9 FFFA               DBEQ    D1,LPRDIN
020DC6  6622                    BNE.S   EXPRDI
020DC8  14FC 0050               MOVE.B  #'P',(A2)+
020DCC  14BC 0043               MOVE.B  #'C',(A2)
020DD0  102B 0001               MOVE.B  01(A3),D0
020DD4  4880                    EXT.W   D0
020DD6  48C0                    EXT.L   D0
020DD8  D08B                    ADD.L   A3,D0
020DDA  2F09                    MOVE.L  A1,-(A7)
020DDC  43FA 0292               LEA     ABER3+20(PC),A1
020DE0  12FC 003D               MOVE.B  #'=',(A1)+
020DE4  6100 01C2               BSR     DIST8B
020DE8  225F                    MOVEA.L (A7)+,A1
020DEA  4E75            EXPRDI: RTS
020DEC
020DEC  610C            BITNRZ: BSR.S   BYTMGS
020DEE  3613                    MOVE.W  (A3),D3
020DF0  0C43 0020               CMPI.W  #$20,D3
020DF4  6400 FBF2               BCC     FEHLER
020DF8  4E75                    RTS
020DFA
020DFA  6100 FD44       BYTMGS: BSR     NXTWRT
020DFE  12FC 0023               MOVE.B  #'#',(A1)+
020E02  12FC 0024               MOVE.B  #'$',(A1)+
020E06  6000 FD62               BRA     WBYTAN
020E0A
020E0A  6100 FD34       WRTMGS: BSR     NXTWRT
020E0E  12FC 0023               MOVE.B  #'#',(A1)+
020E12  12FC 0024               MOVE.B  #'$',(A1)+
020E16  6000 FD4A               BRA     WRT2AN
020E1A
020E1A  12FC 0023       LAWMGS: MOVE.B  #'#',(A1)+
020E1E  6100 FF5A       ABSLAN: BSR     ABSWRT
020E22  6100 FD1C               BSR     NXTWRT
020E26  6000 FD3A               BRA     WRT2AN
020E2A
020E2A  12FC 002C       KCNBWL: MOVE.B  #',',(A1)+
020E2E  7206            CONBWL: MOVEQ   #6,D1
020E30  45FA 0222               LEA     ABER2(PC),A2
020E34  0C1A 002E       LPKTSU: CMPI.B  #'.',(A2)+
020E38  57C9 FFFA               DBEQ    D1,LPKTSU
020E3C  660C                    BNE.S   WGEFUN
020E3E  0C12 0042               CMPI.B  #'B',(A2)
020E42  67B6                    BEQ.S   BYTMGS
020E44  0C12 004C               CMPI.B  #'L',(A2)
020E48  67D0                    BEQ.S   LAWMGS
020E4A  60BE            WGEFUN: BRA.S   WRTMGS
020E4C
```

29.13 Die Übertragung auf andere Computer

Da wohl niemand Programme dieser Länge «von Hand» assemblieren wird, ist die Übertragung nicht schwer.

29.13.1 Realisierung in der Assemblersprache

Hierzu muß man zweierlei beachten:

a) Welche Register sind vom Betriebssystem belegt? Der QL benutzt z. B. A6 als Basisregister; man müßte also A6 als A5 übertragen.
b) Die Input/Output-Organisation anderer Rechner wird sich von der des NDR-Computers erheblich unterscheiden. Hier sind sicherlich Registerinhalte vorübergehend auf den Stapel zu retten. Die Umformung einer Ein- oder Ausgaberoutine kann im Einzelfall schwierig sein. Sie begrenzt sich aber jeweils auf eine einzige Subroutine – und auf die geänderten Sprungweiten reagiert der Assembler von allein.

29.13.2 Übertragung in andere Sprachen

Auch diese Möglichkeit kann im Einzelfall sinnvoll sein. Aus der Fülle der Probleme und Problemlösungen seien wahllos einige herausgegriffen:

a) Verarbeitet die beabsichtigte Sprache auch Hexadezimalzahlen, oder müssen entsprechende Routinen, Prozeduren oder Unterprogramme eigens formuliert werden?
b) Lassen sich auch Unterprogramme mit Namen aufrufen?
c) Wie werden die Speicherzugriffe realisiert? In BASIC mit PEEK. Manche Pascal-Dialekte unterstützen aber den direkten Speicherzugriff nicht.
d) Bis zu wie viele geltende Buchstaben dürfen die Variablen umfassen?
e) Welche Stringoperationen sind verfügbar?
f) Wie sind die Feldvariablen organisiert?

```
020E4C
020E4C  12FC 002C        KUSPAN: MOVE.B  #',',(A1)+
020E50  12FC 0055        USPRAN: MOVE.B  #'U',(A1)+
020E54  12FC 0053                MOVE.B  #'S',(A1)+
020E58  12FC 0050                MOVE.B  #'P',(A1)+
020E5C  4E75                     RTS
020E5E
020E5E  12FC 002C        KSRGAN: MOVE.B  #',',(A1)+
020E62  12FC 0053        SRGRAN: MOVE.B  #'S',(A1)+
020E66  12FC 0052                MOVE.B  #'R',(A1)+
020E6A  4E75                     RTS
020E6C
020E6C  12FC 002C        KCCRAN: MOVE.B  #',',(A1)+
020E70  12FC 0043        CCRRAN: MOVE.B  #'C',(A1)+
020E74  12FC 0043                MOVE.B  #'C',(A1)+
020E78  12FC 0052                MOVE.B  #'R',(A1)+
020E7C  4E75                     RTS
020E7E
020E7E  45FA 01D4        BZUWOL: LEA     ABER2(PC),A2
020E82  7006                     MOVEQ   #6,D0
020E84  0C1A 002E        LBZWL1: CMPI.B  #'.',(A2)+
020E88  57C8 FFFA                DBEQ    D0,LBZWL1
020E8C  661C                     BNE.S   LBZWL4
020E8E  082B 0007 0001           BTST    #7,01(A3)
020E94  6706                     BEQ.S   LBZWL2
020E96  0612 000A                ADDI.B  #10,(A2)
020E9A  4E75                     RTS
020E9C  082B 0006 0001   LBZWL2: BTST    #6,01(A3)
020EA2  6704                     BEQ.S   LBZWL3
020EA4  0612 0015                ADDI.B  #21,(A2)
020EA8  4E75             LBZWL3: RTS
020EAA  6000 FB3C        LBZWL4: BRA     FEHLER
020EAE
020EAE  720F             INVLIS: MOVEQ   #15,D1
020EB0  3414                     MOVE.W  (A4),D2
020EB2  4243                     CLR.W   D3
020EB4  0302             LINVL1: BTST    D1,D2
020EB6  6706                     BEQ.S   NOTSET
020EB8  7C0F                     MOVEQ   #15,D6
020EBA  9C41                     SUB.W   D1,D6
020EBC  0DC3                     BSET    D6,D3
020EBE  51C9 FFF4        NOTSET: DBRA    D1,LINVL1
020EC2  3883                     MOVE.W  D3,(A4)
020EC4  3614             REGLIS: MOVE.W  (A4),D3
020EC6  2A3C 20412044            MOVE.L  #' A D',D5
020ECC  48C3                     EXT.L   D3
020ECE  E09B                     ROR.L   #8,D3
020ED0  610C                     BSR.S   ANALY
020ED2  4845                     SWAP    D5
020ED4  E19B                     ROL.L   #8,D3
020ED6  6106                     BSR.S   ANALY
020ED8
```

ABER1	2102E	EA07FD	20C68	KEA1FF	20C4E	REGLIS	20EC4
ABER2	21054	EA0FFD	20C72	KEA7FD	20C64	REGMEM	2037A
ABER3	2105C	EA0FFF	20C7C	KEAFFD	20C6E	RN	2101C
ABSLAN	20E1E	EAXXEX	20C80	KEAFFF	20C78	RNAN	20B2A
ABSWRT	20D7A	EINLES	20A4E	KEYIN	20056	RNAN2	20B2E
ADRES	20A88	ERSTIN	2038C	KSRGAN	20E5E	SELECT	20AEA
AIDIND	20D28	EXPRDI	20DEA	KUSPAN	20E4C	SELLEX	20B14
AIDIST	20D24	FEHLER	209E8	LANA1	20EEE	SELLOP	20AEC
AIPOST	20D0E	FINIS	20A18	LAUFB1	20F1C	SETCOD	20F4C
AIPRED	20D1A	GEFUN1	20EF8	LAUFEX	20F4A	SHORT	20F88
ANALY	20EDE	GEFUN2	20F08	LAUSG1	20BDE	SPACAN	21004
ARGDIR	20CF0	GR10W	20726	LAUSGB	2004C	SPRUMS	20656
ARINDI	20CF8	GR12W	2087E	LAWMGS	20E1A	SRGRAN	20E62
AUFBAU	20F18	GR13W	20994	LBZWL1	20E84	START	20010
AUSEX	20BEA	GR1W	20076	LBZWL2	20E9C	SUMLOP	20C94
AUSGEB	20BAC	GR2W	200F4	LBZWL3	20EA8	SUP1	203D4
AUSGEX	20BCA	GR4W	20174	LBZWL4	20EAA	SUP2	203D8
B2	21018	GR5W	20206	LEER	20F46	SUP3	203E4
B2DATA	20B34	GR6W	203B6	LGEF11	20EFC	SUP4	20400
B2RN	20B16	GR8RL	205E8	LGEF21	20F0C	SUP5	20732
B3	21019	GR8W	204FE	LGR8RL	205EE	SUP6	20746
B4	2101A	GR9W	2063A	LINVLI	20EB4	SUP7	2074E
B5	2101B	GRENZN	20A2C	LOOP1	2003A	TAB1	2007C
B5RN	20B20	GRUP10	20708	LOOP2	20FBE	TAB10	20766
BIS	21012	GRUP11	2083E	LOPAD1	20A8E	TAB12	208DA
BITNRZ	20DEC	GRUP12	20864	LPKTSU	20E34	TAB13	209AE
BUFER	21022	GRUP13	20976	LPRDIN	20DBE	TAB2	20100
BYTMGS	20DFA	GRUP14	209D2	MEMREG	2036E	TAB4	20194
BZUWOL	20E7E	GRUP2	200DC	MNE8NZ	2060E	TAB5	20242
CCRRAN	20E70	GRUP3	20124	MNEGNZ	20B8A	TAB5A	20394
CDWEIT	205C0	GRUP4	20148	MNERAN	20B7C	TAB6	20414
CHARAN	20FF4	GRUP5	201E8	MODUS	20CA0	TAB8	2051E
CKSRCR	20C1A	GRUP5A	20332	NCOUNT	205BC	TAB8A	205C8
CNKCCR	20C1C	GRUP6	2039C	NIBAS	20B9A	TAB9	2066C
CODLIS	21020	GRUP7	20498	NIBEX	20BAA	TBIS	20A4A
CONBWL	20E2E	GRUP8	204D6	NIBMGS	20B92	TCONCO	20F68
CONCOD	20F56	GRUP8A	2055A	NICHTD	20D52	TEAXX	20CB0
CONKSR	20C28	GRUP8B	20582	NICHTW	20D74	TEXTAN	20B80
DATRAN	20B36	GRUP9	2061C	NOT8RX	2060C	TFEHL	20A0A
DECODE	20062	HXDUMP	20FB4	NOTADX	20C92	TIMBWL	20C0A
DEFJOB	20A1A	IMMBWL	20BF0	NOTL	2035A	TJOB	20A28
DEPA7	2100A	INVLIS	20EAE	NOTSET	20EBE	TVON	20A46
DISASS	20030	JOB	2101D	NOTSHT	20664	USPRAN	20E50
DISAUS	20BDA	KAIDIS	20D20	NUR1	20F42	VON	2100E
DIST16	20FA2	KAIPRE	20D16	NUR12	20F3C	WBYTAN	20B6A
DIST8B	20FA8	KAIPST	20D0A	NW	2101E	WGEFUN	20E4A
DRGDIR	20CE4	KARDIR	20CEC	NXTJOB	20016	WORTAN	20B50
EA01F4	20C34	KCCRAN	20E6C	NXTWRT	20B40	WRT2AN	20B62
EA01FC	20C3E	KCNBWL	20E2A	OC	21016	WRT2MS	20B5E
EA01FD	20C48	KDRDIR	20CE0	OPCODE	20AB0	WRTMGS	20E0A
EA01FF	20C52	KEA1F4	20C30	PRDIND	20DB4	ZEILOP	20FD6
EA07EC	20C58	KEA1FC	20C3A	PRELDI	20D86		
EA07F4	20C5E	KEA1FD	20C44	PRIAUS	20BCC		

Die Symboltafel

```
020ED8  133C 0020            MOVE.B  #' ',-(A1)
020EDC  4E75                 RTS
020EDE
020EDE  45FA 0142      ANALY: LEA     BUFER(PC),A2
020EE2  429A                  CLR.L   (A2)+
020EE4  429A                  CLR.L   (A2)+
020EE6  429A                  CLR.L   (A2)+
020EE8  45FA 0137             LEA     BUFER-1(PC),A2
020EEC  78FF                  MOVEQ   #-1,D4
020EEE  5204           LANA1: ADDQ.B  #1,D4
020EF0  4A03                  TST.B   D3
020EF2  6724                  BEQ.S   AUFBAU
020EF4  E303                  ASL.B   #1,D3
020EF6  64F6                  BCC.S   LANA1
020EF8  45EA 0001      GEFUN1: LEA    01(A2),A2
020EFC  5212           LGEF11: ADDQ.B #1,(A2)
020EFE  4A03                  TST.B   D3
020F00  6716                  BEQ.S   AUFBAU
020F02  E303                  ASL.B   #1,D3
020F04  6402                  BCC.S   GEFUN2
020F06  60F4                  BRA.S   LGEF11
020F08  45EA 0001      GEFUN2: LEA    01(A2),A2
020F0C  5212           LGEF21: ADDQ.B #1,(A2)
020F0E  4A03                  TST.B   D3
020F10  6706                  BEQ.S   AUFBAU
020F12  E303                  ASL.B   #1,D3
020F14  65E2                  BCS.S   GEFUN1
020F16  60F4                  BRA.S   LGEF21
020F18  45FA 0108      AUFBAU: LEA    BUFER(PC),A2
020F1C  101A           LAUFB1: MOVE.B (A2)+,D0
020F1E  672A                  BEQ.S   LAUFEX
020F20  12C5                  MOVE.B  D5,(A1)+
020F22  1284                  MOVE.B  D4,(A1)
020F24  0619 0030             ADD.B   #$30,(A1)+
020F28  0C00 0001             CMPI.B  #1,D0
020F2C  6714                  BEQ.S   NUR1
020F2E  12FC 002D             MOVE.B  #'-',(A1)+
020F32  12C5                  MOVE.B  D5,(A1)+
020F34  D800                  ADD.B   D0,D4
020F36  1284                  MOVE.B  D4,(A1)
020F38  0619 002F             ADDI.B  #$2F,(A1)+
020F3C  12FC 002F      NUR12: MOVE.B  #'/',(A1)+
020F40  6004                  BRA.S   LEER
020F42  5204           NUR1:  ADDQ.B  #1,D4
020F44  60F6                  BRA.S   NUR12
020F46  D81A           LEER:  ADD.B   (A2)+,D4
020F48  60D2                  BRA.S   LAUFB1
020F4A  4E75           LAUFEX: RTS
020F4C
```

```
020F4C  163A 00C8        SETCOD: MOVE.B  0C(PC),D3
020F50  0243 000F                ANDI.W  #$0F,D3
020F54  D643                     ADD.W   D3,D3
020F56  49FA 0010        CONCOD: LEA     TCONCO(PC),A4
020F5A  49F4 3000                LEA     00(A4,D3.W),A4
020F5E  12DC                     MOVE.B  (A4)+,(A1)+
020F60  12DC                     MOVE.B  (A4)+,(A1)+
020F62  43FA 00F8                LEA     ABER3(PC),A1
020F66  4E75                     RTS
020F68  5420462048494C   TCONCO: DC.B    'T F HILS'
020F6F  53
020F70  434343534E4545           DC.B    'CCCSNEEQ'
020F77  51
020F78  5643565350 4C4D          DC.B    'VCVSPLMI'
020F7F  49
020F80  47454C5447544C           DC.B    'GELTGTLE'
020F87  45
020F88
020F88  137C 002E FFFA  SHORT:   MOVE.B  #'.',-6(A1)
020F8E  137C 0053 FFFB           MOVE.B  #'S',-5(A1)
020F94  102B 0001                MOVE.B  01(A3),D0
020F98  4880                     EXT.W   D0
020F9A  48C0                     EXT.L   D0
020F9C  5480                     ADDQ.L  #2,D0
020F9E  D08B                     ADD.L   A3,D0
020FA0  6006                     BRA.S   DIST8B
020FA2  200B             DIST16: MOVE.L  A3,D0
020FA4  D07A 0078                ADD.W   NW(PC),D0
020FA8  C149             DIST8B: EXG     A0,A1
020FAA  4EB9 0000164A            JSR     @PRINT8X
020FB0  C348                     EXG     A1,A0
020FB2  4E75                     RTS
020FB4                   * ==============================
020FB4  4EB9 00003FEC    HXDUMP: JSR     @CLRSCREEN
020FBA  267A 0052                MOVEA.L VON(PC),A3
020FBE  247A 0052        LOOP2:  MOVEA.L BIS(PC),A2
020FC2  B7CA                     CMPA.L  A2,A3
020FC4  6E00 F090                BGT     KEYIN
020FC8  6100 FABE                BSR     ADRES
020FCC  43FA 0094                LEA     ABER1+52(PC),A1
020FD0  7207                     MOVEQ   #7,D1
020FD2  47EB FFFE                LEA     -2(A3),A3
020FD6
```

```
020FD6  6100 FB68       ZEILOP: BSR     NXTWRT
020FDA  163A 0042               MOVE.B  NW(PC),D3
020FDE  6114                    BSR.S   CHARAN
020FE0  163A 003D               MOVE.B  NW+1(PC),D3
020FE4  610E                    BSR.S   CHARAN
020FE6  51C9 FFEE               DBRA    D1,ZEILOP
020FEA  6100 FBC0               BSR     AUSGEB
020FEE  47EB 0002               LEA     02(A3),A3
020FF2  60CA                    BRA.S   LOOP2
020FF4  0C03 001F       CHARAN: CMPI.B  #$1F,D3
020FF8  6F0A                    BLE.S   SPACAN
020FFA  0C03 007E               CMPI.B  #$7E,D3
020FFE  6204                    BHI.S   SPACAN
021000  12C3                    MOVE.B  D3,(A1)+
021002  4E75                    RTS
021004  12FC 0020       SPACAN: MOVE.B  #' ',(A1)+
021008  4E75                    RTS
02100A                  * ================================
02100A                  DEFA7:  DS.L    1
02100E                  VON:    DS.L    1
021012                  BIS:    DS.L    1
021016                  OC:     DS.W    1
021018                  B2:     DS.B    1
021019                  B3:     DS.B    1
02101A                  B4:     DS.B    1
02101B                  B5:     DS.B    1
02101C                  RN:     DS.B    1
02101D                  JOB:    DS.B    1
02101E                  NW:     DS.W    1
021020                  CODLIS: DS.W    1
021022                  BUFER:  DS.B    12
02102E                  ABER1:  DS.W    19
021054                  ABER2:  DS.W     4
02105C                  ABER3:  DS.W    16
02107C
02107C                          END
```

30
Nachlese der nicht privilegierten Befehle

In diesem Kapitel sollen, bevor wir uns mit den privilegierten Befehlen des Betriebssystems (Supervisor-Ebene) befassen, in loser Folge einige besonders interessant erscheinende Befehle der User-Ebene nachgetragen werden.

30.1 Bit/Byte-Abfrage und Bit/Byte-Manipulation

Die Befehle BCHG, BCLR und BSET prüfen ein Bit des adressierten Operanden, dessen Position (Ordnungszahl) entweder als Konstante gegeben oder in ein Datenregister eingetragen ist. Dieser Prüfung entsprechend wird das Z-Flag gesetzt oder gelöscht. Bitte erinnern Sie sich: Z=1 besagt, daß eine Null festgestellt wurde. Unterschiede:

BCLR löscht das spezifizierte Bit anschließend,
BSET setzt es, und
BCHG invertiert es (change).
(BTST beläßt es beim Test und verändert das Bit nicht.)
TAS (test and set) prüft das adressierte Byte auf Null und Vorzeichen und setzt N- und Z-Flag entsprechend.
Scc ist das ungefähre Gegenstück zu TAS: Abhängig vom CCR wird ein Byte entweder auf 00 oder auf FF gesetzt.

30.2 Weiteres zum Stapel

PEA (push effective address) berechnet eine effektive Adresse und legt sie auf dem Stapel ab; damit kann man z.B. die Rückkehr aus einer Subroutine zu einem neuen Ziel umlenken.

P.30.1 rettet den offiziellen Stackpointer nach A4 und richtet vorübergehend einen eigenen A7-Stapel ein, um mit ihm experimentieren zu können.

```
    3                           ; P.30.1: Der Befehl PEA
    4
    5 000000 284F               start:    move.l  a7,a4
    6 000002 4FFA0022                     lea     stapel+8(pc),a7
    7 000006 6108               loop:     bsr.s   xpeax
    8 000008 60FC                         bra.s   loop
    9
   10 00000A 2E4C               lorley:   move.l  a4,a7
   11 00000C 7200                         moveq   #0,d1
   12 00000E 4E75               end:      rts
   13
   14 000010 3EBCAFFE           xpeax:    move.w  #$affe,(a7)
   15 000014 47FAFFF0                     lea     loop(pc),a3
   16 000018 486B0004                     pea     04(a3)
   17 00001C 4E75                         rts
   18
   19 00001E 0000               stapel:   blk.w   4,0
   20
   21 000026                              end
No Errors

--SYMBOL TABLE--

END.......0000000E+ LOOP......00000006+ LORLEY....0000000A+
STAPEL....0000001E+
START.....00000000+ XPEAX.....00000010+

035FFC 284F4FFA 00226108 60FC2E4C 72004E75 (OOz."a.`!.Lr.Nu
03600C 3EBCAFFE 47FAFFF0 486B0004 4E750003 ><^Gz.pHk..Nu..
03601C 6006AFFE 600478DD 00007BE3 00000036 `./^`.x]..{c...6
03602C 7FFFBF92 00000053 00000000 00001034 ..?....S.......4
03603C 0000AF1F 00000000 00000000 00000000 ../.............
03604C 00000000 0000D868 00000000 0000D878 .......Xh.......Xx
03605C 00000000 00000000 00000000 000031A7 ..............1'
03606C 00000048 0000004F 00000000 0000004E ...H...O.......N
```

P.30.1 Eine einfache Anwendung des Befehls PEA

Vor Rückkehr nach Amiga- bzw. Seka-DOS muß A7 wiederhergestellt werden. Der Nicht-Amiga-User denke daran, daß er sicherlich MOVEA <ea>,An statt MOVE (ea>,An notieren muß. Auch die Rücksprungkonvention D1 = 0 sieht bei ihm wahrscheinlich anders aus.

Normalerweise müßte unser Programm bei LOOP in einer endlosen Schleife hängen bleiben, doch die Subroutine xpeax verstellt den Rücksprung: Zunächst wird die von A7 angezeigte Adresse mit einem Urwaldbewohner markiert. Dann lädt LEA A3 mit der absoluten Adresse von loop = 035FFC + 000006, und PEA macht daraus 036002 + 04 = 036006, also die mit Label lorley markierte Adresse, und legt diese auf dem (wohlgemerkt auf unserem momentanen) Stapel ab. Sie werden diese Adresse sicherlich im Speicherauszug finden. Offensichtlich legt PEA, wie in diesem Zusammenhang üblich, mit Predekrement ab. So findet denn das folgende RTS eine zum Programmausgang führende Rücksprungadresse vor.

A.30.1:

a) Vollziehen Sie die PEA-Rechnung mit relativen Adressen nach!
b) Woher rührt der Stapeleintrag 6004?

LINK ermöglicht die Erweiterung eines Stapels bzw. die Einrichtung eines eigenen Stapels für ein Unterprogramm. UNLK stellt nach Rückkehr aus der Subroutine den alten Zustand wieder her. Zu Syntax und Befehlsablauf: LINK An,#Konst legt zuerst den Inhalt des Rahmenzeigers (frame pointer) An auf dem Stapel ab. Der FP übernimmt dann den Inhalt des Stackpointers A7. Abschließend wird der Inhalt von A7 um den Betrag der vorzeichenbehafteten Konstante vermehrt (bzw. vermindert). UNLK An stellt den alten Zustand wieder her. Bei C. VIEILLEFOND (s. Literaturverzeichnis) findet sich der Hinweis, daß man dieses Befehlspaar z.B. zur Realisierung lokaler Variablen in Funktionen und Prozeduren höherer Sprachen benutzen kann. Unser Programmbeispiel ist nahezu trivial. Bitte versuchen Sie, folgende Fragen zu beantworten:

A.30.2:

a) Welches ist die Basisadresse des Programms?
b) Zu welchen Adressen kehrt der Programmlauf nach den Unterprogrammaufrufen zurück?
c) Wieso wird der Stapel de facto nur in Zwölferschritten erweitert, wo doch je 20 Bytes gelinkt werden?

```
   3                                ; P.30.2: LINK-Demo
   4
   5 000000 284F                    start:   move.l   a7,a4
   6 000002 721E                             moveq    #30,d1
   7 000004 4FFA0040                         lea      stapel(pc),a7
   8 000008 425F                    loop:    clr.w    (a7)+
   9 00000A 51C9FFFC                         dbra     d1,loop
  10
  11 00000E 4FFA0040                         lea      stapel+10(pc),a7
  12 000012 267C12345678                     move.l   #$12345678,a3
  13 000018 6106                             bsr.s    subru1
  14
  15 00001A 2E4C                             move.l   a4,a7
  16 00001C 7200                             moveq    #0,d1
  17 00001E 4E75                   end:      rts
  18
  19 000020 4E530014               subru1:   link     a3,#20
  20 000024 6104                             bsr.s    subru2
  21 000026 4E5B                             unlk     a3
  22 000028 4E75                             rts
  23
  24 00002A 4E530014               subru2:   link     a3,#20
  25 00002E 6104                             bsr.s    subru3
  26 000030 4E5B                             unlk     a3
  27 000032 4E75                             rts
  28
  29 000034 4E530014               subru3:   link     a3,#20
  30 000038 6104                             bsr.s    subru4
  31 00003A 4E5B                             unlk     a3
  32 00003C 4E75                             rts
  33
  34 00003E 3F7CBADE0008           subru4:   move.w   #$bade,08(a7)
  35 000044 4E75                             rts
  36
  37 000046 0000                   stapel:   blk.w    30,0
  38
  39 000082                                  end
No Errors
```

```
--SYMBOL TABLE--

END.......0000001E+ LOOP......00000008+ STAPEL....00000046+
START.....00000000+
SUBRU1....00000020+ SUBRU2....0000002A+ SUBRU3....00000034+
SUBRU4....0000003E+
0361CC 284F721E 4FFA0040 425F51C9 FFFC4FFA (Or.Oz.@B_QI.!Oz
0361DC 00402670 12345678 61062E4C 72004E75 .@&!.4Vxa..Lr.Nu
0361EC 4E530014 61044E5B 4E754E53 00146104 NS..a.N[NuNS..a.
0361FC 4E5B4E75 4E530014 61044E5B 4E753F7C N[NuNS..a.N[Nu?!
03620C BADE0008 4E750000 12345678 000361E6 :^..Nu...4Vx..af
03621C 00000000 00036214 000361F2 00000000 ......b...ar....
03622C 00036220 000361FC 00000000 00000000 ..b ..a!........
03623C 00036206 00000000 BADE0000 00000000 ..b.....:^......
```

P.30.2 Eine Anwendung der Befehle LINK und UNLK

d) Welcher gravierende Einwand läßt sich gegen den Programmansatz vorbringen?
e) Kann man die Struktur mit einem einzigen RTS à la P.10.2 nachvollziehen?

30.3 Datentransfer von der und zur Peripherie (MOVEP)

Weil hier Hardwarefragen bloß am Rande interessieren, sei nur beiläufig erwähnt, daß der M68000 das ihm zugeordnete Bussystem synchron und asynchron betreiben kann. Speicher und periphere 16-bit-Bausteine werden asynchron mit DS/AS/DTACK-Handshake bedient, 8-bit-Peripheriebausteine synchron. Hier sei seines Namens wegen der PTM MC6840 genannt, ein «dreifach programmierbarer Timer». Wie der Name sagt, muß ein solcher Baustein – wie mehrere andere auch –, bevor er seine Aufgabe erfüllen kann, programmiert werden, indem man ihm in einer bestimmten Reihenfolge über bestimmte Adressen bestimmte 8-bit-Muster, also Bytes, zuführt.

Hierfür wird der Befehl MOVEP eingesetzt; er überträgt .W- und .L-Daten byteweise von/zur Peripherie; es ist der einzige Befehl, der .W- oder .L-Daten auf ungeraden Adressen lesen oder schreiben kann.

31
Ausnahmen vom normalen Programmablauf (I)

Ein unbefangener Computerbenutzer mag annehmen, daß sein Programm zügig Befehl um Befehl abgearbeitet wird; das ist jedoch nur selten der Fall. Eine Uhr kann z.B. dadurch realisiert sein, daß ein Timer-Baustein alle Sekunden einen Impuls abgibt, der den normalen Programmablauf unterbricht (Interrupt), die h-m-s-Speicherstelle um eins weiterzählt und dann den Programmablauf wieder freigibt.

Ein Prozessor kann mehrere Benutzerprogramme nur dadurch scheinbar gleichzeitig ablaufen lassen, daß er mal das eine, dann das andere ein bißchen laufen läßt.

Es gibt mehrere derartige Ausnahmen vom normalen Programmablauf; ihnen sollen die folgenden Kapitel gewidmet sein.

31.1 Das Statusregister

Das Statusregister umfaßt, wie wir bereits wissen, 16 bit. Auf sein LSB, das CCR, braucht nicht mehr eingegangen zu werden. Von den acht Bits des MSBs sind wie beim CCR nur fünf belegt, wenn auch mit völlig unterschied-

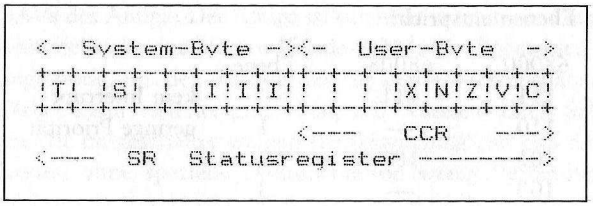


Bild 31.1 Das Statusregister

licher Bedeutung: Die Bits 8 ... 10 bilden die Interrupt-Maske, auf die wir schon bald eingehen werden. Bit13 ist das Supervisor-Bit; befindet sich der Prozessor im Supervisor-Modus, ist dieses Bit gesetzt, sonst gelöscht. Bit 15 ist das Trace-Bit; ist es gesetzt, arbeitet der Prozessor im Single-step-Modus einzelne Befehle ab; man kann damit – wie z.B. mit dem NDR-Computer – hervorragend Programme testen und ggf. Fehler aufspüren.

A.31.1: Schreiben Sie ein Programm, das diese Flags abfragt.

31.2 Die Interrupt-Ebenen

Nehmen wir einmal an, ein Computer steuere einen Plotter. Durch eine Unachtsamkeit des Programmierers fährt der Zeichenstift über den Rand hinaus, der X-Motor bekommt dennoch weitere Impulse, und das Gestänge verbiegt sich. Man kann das u.a. dadurch verhindern, daß am Anschlag ein Schalter betätigt wird, der einen Interrupt auslöst, d.h. den normalen Programmablauf unterbricht, und Fehlermeldungen oder Korrekturen auslöst.

Nehmen wir ferner an, derselbe Computer steuere im Multitasking-Betrieb auch noch die Herstellung von Dynamit. Ein Temperaturfühler meldet, daß die Temperatur im Reaktionsgefäß den zugelassenen Höchstwert mit «Tendenz steigend» erreicht hat. Auch hier könnte ein Interrupt Abhilfe schaffen; aus verständlichen Gründen müßte er Vorrang vor dem angenommenen Plotter-Interrupt haben.

Es gibt aber auch Bausteine (z.B. Timer), die periodisch Interrupts zur Steuerung interner Abläufe (z.B. Uhr, Scheduler) auslösen.

Der M68000 hat drei Interrupt-Eingänge, $\overline{\text{IPL0}}$, $\overline{\text{IPL1}}$ und $\overline{\text{IPL2}}$. Mit drei Bits lassen sich bekanntlich acht verschiedene Zahlen darstellen; dementsprechend sind acht verschiedene Interrupt-Ebenen denkbar.

Beim M68008 sind $\overline{\text{IPL0}}$ und $\overline{\text{IPL2}}$ intern miteinander verbunden, was vier möglichen Ebenen entspricht:

68000	68008	Ebene	
111	111	0	kein Interrupt
110	—	1	geringe Priorität
101	101	2	
100	—	3	
011	—	4	

010	010	5	
001	—	6	
000	000	7	höchste Priorität

Abgesehen von den unterschiedlichen Prioritätsrängen gibt es weitere Feinunterscheidungen, so sind die Prioritätsebenen 1 – 6 pegelgetriggert, reagieren also auf anliegende Nullpotentiale; Ebene 7 ist hingegen flankengetriggert, reagiert also auf den Potentialabfall. Wichtiger als das ist für den Programmierer, daß Interrupts der Ebene 7 nicht maskierbar (NMI, non maskable interrupt) sind, während die der Ebenen 1 ... 6 vor der Ausführung mit der in Abschnitt 31.1 erwähnten Interrupt-Maske verglichen werden: Angenommen, die Interrupt-Maske enthalte 101 = 5, dann kann neben dem stets verfügbaren NMI (Ebene 7) nur noch ein Interrupt der Ebene 6 (weil > 5) ausgeführt werden.

A.31.2:

a) Setzen Sie die Interrupt-Maske Ihres Computers auf 2.
b) Schreiben Sie ein Programm, das primär Veränderungen der Interrupt-Maske durch das Betriebssystem aufspürt.

Hat der Prozessor einen Interrupt erkannt, gibt es im wesentlichen zwei Möglichkeiten, die nur kurz erwähnt werden können. In Abhängigkeit von den Signalleitungen $\overline{\text{VPA}}$ und $\overline{\text{DTACK}}$ wird entweder ein Autovektor- oder ein Non-Autovektor-Interrupt ausgeführt: Im zweiten Falle wird die Vektoradresse von dem auslösenden peripheren Baustein geliefert, im ersten, dem wir kurz folgen wollen, sind diese Vektoren unter den Adressen 064 (Interrupt-Autovektor der Ebene 1) ... 07C (Interrupt-Autovektor der Ebene 7) im Betriebssystem (meist ROM) zu finden. Bild 31.2 zeigt diese Vektoren für den QL, den NDR-Computer, beide mit CPU M68008, und den Amiga mit CPU 68000.

Zu den IAVs des Amiga: Der Amiga ist offenbar weithin ein Cleancomputer mit einem vielleicht nur kleinen Urlade-ROM. Im allgemeinen sind drei Ladevorgänge nötig, um den Rechner bis zur Betriebsbereitschaft «hochzufahren». Dabei kann vielerlei geschehen, z.B. können durch Softschalter Speicherbereiche umgeschaltet werden (bankswitching), so daß die angegebenen Adressen ohne spezielle Dokumentation wenig Verbindliches aussagen.

	Amiga	NDR	QL
Ebene 0 000060:	FC0750	003336	00005E
Ebene 1 000064:	FC0A96	003336	00005E
Ebene 2 000068:	FC0AEA	008000	000340
Ebene 3 00006C:	FC0B1C	003336	00005E
Ebene 4 000070:	FC0B74	003336	00005E
Ebene 5 000074:	FC0C02	008006	00005E
Ebene 6 000078:	FC0C48	003336	00005E
Ebene 7 00007C:	FC0C8E	00800C	000036

Bild 31.2 Interrupt-Autovektoren

An den IAVs des NDR-Computers fällt auf, daß den beim M68008 gültigen Ebenen Adressen im RAM-Bereich (ab $8000) entsprechen, was es ermöglicht, je nach Vorhaben mal diese und mal jene Adresse einzuschreiben, was an sich zum Experimentieren verlockt, doch soll hier auf auslösende Hardware-Manipulationen nicht eingegangen werden.

In $8000 und $8006 sind normalerweise Sprünge – auch nach $3336 – und in $800C ein Sprung nach $2FBA eingetragen. Bild 31.3 vermittelt einen kleinen Eindruck vom Beginn der beiden relevanten Interrupt-Routinen:

Unser Bild zeigt aber noch etwas mehr. Wenn man unbefangen irgendwo zu disassemblieren beginnt, kann man in den April geschickt werden; beginnt man z. B. bei 2FA8, trifft man zunächst auf Befehle, die den Programmablauf nicht umlenken, und landet beim schon mehrfach beobachteten 3336. Wie Teil b) zeigt, steht aber in 2FBA ein RTE (return from exception), ein dem RTS ähnlicher Befehl, auf den wir bald genauer eingehen werden.

Bei 003336 stehen zunächst mehrere Wertzuweisungen nach A6, gefolgt von unbedingten Sprüngen. Beim Auftreten eines Interrupts wird hier A6 auf den Text ‹Interrupt› gerichtet, der zusammen mit einer Registerinhaltsanzeige ausgegeben wird, nachdem ab 00338C diese Inhalte zwischengespeichert wurden.

```
a) ...   .... .... ....   .......
002FA4   6000 F87C        BRA      $002822
002FA8   0000 0000        ORI.B    #$00,D0
002FAC   0000 0000        ORI.B    #$00,D0
002FB0   0000 0000        ORI.B    #$00,D0
002FB4   0000 0000        ORI.B    #$00,D0
002FB8   0000 4E73        ORI.B    #$73,D0
002FBC   4EF9 0000 3336   JMP      $003336

b) ...   .... .... ....   ........
002FBA   4E73             RTE

c) ...   .... .... ....   ........
003336   4DFA 04F6        LEA      $04F6(PC),A6   =00382E
00333A   6000 0050        BRA      $00338C
00333E   4DFA 04F8        LEA      $04F8(PC),A6   =003838
003342   6000 0042        BRA      $003386

d) ...   .... .... ....   ........
003386   DFFC 0000 0008   ADDA.L   #$00000008,A7
00338C   4BFA CC72        LEA      $CC72(PC),A5   =000000
003390   DBFA D072        ADDA.L   $D072(PC),A5   =000404
001194   48ED FFFF 0178   MOVEM.L  D0-D7/A0-A7,$0178(A5)
.......  .... .... ....   ........
```

Bild 31.3 Interruptverlauf beim NDR-Computer

31.3 Die Interrupt-Ebenen des QL

Bild 31.2 zeigt, daß beim QL sechs Interruptvektoren (nur einer von diesen gilt wirklich) nach 005E führen. Dort steht RTE, wie eben beim NDR-Computer erwähnt.

Ebene 2

Ab 000340 werden zunächst einige Register auf den Stapel gerettet. Anschließend wird A6 auf die Basisadresse $28000 gerichtet. (Von hier ab sind beim QL [engl. Version] die Systemvariablen im Speicher zu finden, z.B. der Anfang der Liste der momentan zu betreuenden Jobs, der Anfang der Liste der Ein-/Ausgabekanäle, die momentane Zufallszahl u.v.a.m.) Nun wird ein Byte aus $018021 nach D7 geholt; nach NAYLOR und ROGERS (s.

Literaturverzeichnis) ist das eine Hardware-Input/Output-Adresse. QL-Besitzer, die über einen Disassembler verfügen, werden nachvollziehen können, daß anschließend einzelne Bits von D7 geprüft werden:

Wenn Bit0 = 1, dann weiter bei 0029D6;
wenn Bit1 = 1, dann weiter bei 002BE6;
wenn Bit2 = 1, dann weiter bei 002BF2 usw.

Hinter dieser Abfrage kann vielerlei stecken, z.B.:

Steckt eine Cartridge im Mikrodrive?
Hat die Tastatur eine Eingabe registriert?
Muß jetzt ein anderer Job Rechenzeit erhalten?

Nach PENNELL hat ein *Ebene-2-Interrupt* folgende Aufgaben zu lösen:

Tastaturabfrage (Scanning);
Ein- und Ausgabe über RS 232;
Schreiben/Lesen auf/von Mikrodrives;
Verwaltung externer Interrupts;
Blinken des Cursors und
Rechenzeitzuweisung durch den Scheduler.

Auf den letzten Punkt gehen wir bei der Besprechung des Multitasking näher ein.

Ebene-7-Interrupt

In 000036 wird die Subroutine 000050 aufgerufen, die mit BRA 00013A beginnt; dort steht

```
00013A 4AB9 0002 8050    TST.L $028050,
```

es wird also eine Systemvariable abgefragt. Bei A. DICKENS finden wir diese als «pointer to trap redirektion table» aufgeführt. Es ist offenbar an der Zeit, uns mit den Traps vertraut zu machen.

32
Ausnahmen vom normalen Programmablauf (II): Die TRAP-Befehle

Ein wesentliches Merkmal des M68000 ist seine Fähigkeit, Mehrbenutzersysteme zu verwalten. Über den Amiga läßt sich dazu im Augenblick nur sagen, daß sich unter den mitgelieferten Demos Programme befinden, die offenbar im Multitasking ablaufen: In verschiedenen Fenstern tut sich allem Anschein nach unabhängig voneinander Verschiedenes – scheinbar gleichzeitig. Auf dem QL werden wir mehrere Programme in willkürlicher Zusammenstellung unabhängig voneinander laufen lassen.

Beim Multitasking sind folgende Probleme zu bewältigen: Die einzelnen Programme (bzw. die verschiedenen Benutzer) dürfen sich nicht gegenseitig stören. Es wäre unpraktikabel, für jeden User die Standardroutinen des Betriebssystems neu zu formulieren; sie sind nur einmal implementiert und können von allen Anwendern benutzt werden, jedoch auch ohne gegenseitige Störung.

32.1 User- und Supervisor-Modus

Der M68000 bewältigt die Probleme des Multitasking durch eine Arbeit auf zwei Ebenen:

Ist das S-Bit (s. Bild 31.1) gesetzt, befindet sich der Prozessor im Supervisor-Modus, in dem alle Befehle zugelassen sind, auch die «privilegierten». Der Prozessor befindet sich nun in einem Ausnahmezustand (exception):

Das Betriebssystem (der Supervisor) unterhält einen eigenen Stapel und hat seinen eigenen Stapelzeiger. Auch er heißt A7. Bitte betrachten Sie noch einmal Bild 2.1 (die Register des M68000); Sie sehen dort zwei Stackpointer: A7 (User-Stackpointer, USP) und A7' (Supervisor-Stackpointer, SSP). Das Hochkomma ist nicht sehr hilfreich und wird in der Praxis wohl auch kaum benutzt; die praktische Unterscheidung sieht so aus:

User-Modus

Der Anwender kann nur über seinen Stapel verfügen. Er erreicht ihn unter dem Namen A7.

Supervisor-Modus

Unter dem Namen A7 erreicht das Betriebssystem seinen Stapel. Als übergeordnete Instanz hat es aber auch Zugriff auf den User-Stapel. Da der Name A7 nun anderweitig belegt ist, heißt der User-Stackpointer jetzt USP.

A.32.1: Das Betriebssystem soll den Rücksprung aus einer momentan unterbrochenen User-Subroutine nach $123456 umlenken. Wie lauten die beiden wesentlichen Befehle?

Auf die privilegierten Befehle gehen wir später ein und wenden uns nun der Frage zu, wie man denn eigentlich in den Supervisor-Modus gelangt.

32.2 Aufgabe und Organisation der TRAPs

Laut Langenscheidt-Wörterbuch bedeutet trap «Falle, Wurfmaschine (bes. beim Tontaubenschießen)». Man kann sich also vorstellen, entweder durch eine Falltür oder mit einer Schleuder in das Betriebssystem zu gelangen, genauer: Es gibt 16 derartige Falltüren; nicht alle geben in allen 68000-Computern unterirdische Gänge (bezogen auf die User-Ebene) frei.

Befehl	Vektor#	Adresse	Sprungziel
TRAP #0	#20	000080:	000308
TRAP #1	#21	000084:	00030C
TRAP #2	#22	000088:	000312
TRAP #3	#23	00008C:	000318
TRAP #4	#24	000090:	00031E
TRAP #5	#25	000094:	000038
TRAP #6	#26	000098:	00003A
.... ..	...	.....	.....
TRAP #F	#2F	0000BC:	00004C

Bild 32.1 Die TRAP-Vektoren beim QL (engl. Version)

Diese Zugänge zur Supervisor-Ebene erreicht man mit den TRAP-Befehlen. Sie sind beim QL letztlich so geordnet:

TRAP #1 – Manager traps
Man erreicht z.B. mit der Funktionsauswahlnummer

D0 =	0	MT_INF	Information über z.B. lfd. Job
D0 =	1	MT_CJOB	Neuen Job einrichten (create)
D0 =	4	MT_RJOB	Entfernt (remove) Job aus Speicher
D0 =	6	MT_FREE	Sucht längsten zusammenhängenden freien Speicherraum
D0 =	8	MT_SUSJB	Suspendiert einen Job, unterbricht
D0 =	$1A	MT_LXINT	Linkt eine externe Interrupt-Routine

TRAP #2 – Input/Output-Zuteilung

D0 =	1	IO_Open	Öffnet einen Kanal
D0 =	2	IO_CLOSE	Schließt einen Kanal
D0 =	4	IO_DELET	Löscht ein File (z.B. eine Bandaufzeichnung)

TRAP #3 – Input/Output-Ausführung

D0 =	1	IO_FBYTE	Nimmt ein Byte aus einem Kanal auf (fetch a byte)
D0 =	7	IO_SSTRG	Stringausgabe (send a string of bytes)

TRAP #3 – Bildschirmausgabekontrolle

D0 =	$A	SD_PXENQ	Bestimmt Fensterabmessungen und Cursorposition in Pixel
D0 =	$B	SD_CHENQ	Bestimmt Fensterformat und Cursor durch Schreibpositionen
D0 =	$18	SD_SCROL	Scrollt den Bildschirminhalt
D0 =	$2D	SD_SETSZ	Bestimmt die Schriftgröße (set size)

TRAP #4 – ist dem BASIC-Interpreter vorbehalten

TRAP #5 ... #$F werden von QDOS nicht benutzt.

Die einzelnen Funktionen werden durch Mehrfachauswahl bestimmt, wie sie z.B. im großen Sortierprogramm vorgestellt wurde.

32.3 Vektorisierte Routinen (II)

Unter den spontan herausgegriffenen Beispielen sind Ihnen sicherlich die TRAP-Routinen zur Festlegung der Fensterabmessungen aufgefallen; ähnliches haben wir doch mit UT_SCR bewirkt. Das Besondere derartiger Routinen liegt gar nicht in der Vektorisierung; das ist lediglich eine Adressiertechnik, bei der man die in eine Tabelle eingeschriebene Einsprungadresse einer Routine als Vektor bezeichnet. Man holt in der Regel diese Einsprungadresse mit einem MOVEA in ein Adreßregister und ruft die Routine dann mit JMP (An) oder JSR (An) auf. Das ist lediglich eine Sonderform der indirekten Adressierung.

Das eigentlich auch gar nicht so Besondere einer vektorisierten Routine liegt darin, daß sie a) viele kleine Einzelschritte aneinanderreiht und b) ihren Aufruf von den TRAPs zum vertrauteren JSR usw. vereinfacht. Bild 32.2 stellt auszugsweise UT_SCR dar. Um ein Fenster auf dem Bildschirm zu öffnen, müssen seine Parameter zu einem Definitionsblock zusammengestellt sein, und A1 wird als Pointer auf diesen Block gerichtet. Die Fensterdefinition sieht hier so aus:

(A1)+00.B:	Farbe des Randes
(A1)+01.B:	Breite des Randes
(A1)+02.B:	Hintergrundfarbe
(A1)+03.B:	Farbe der Schrift
(A1)+04.W:	Breite des Fensters
(A1)+06.W:	Höhe des Fensters
(A1)+08.W:	X-Abstand des Fensters
(A1)+0A.W:	Y-Abstand des Fensters

In Bild 32.2 läßt sich gut verfolgen, wie diese Vorgaben an verschiedene Einzelaufrufe wie

SD_WDEF:	Fensterdefinition
SD_BORDR:	Randbreite und -farbe setzen
SD_SETPA:	Hintergrundfarbe bestimmen

weitergereicht werden. Bei erfolgreichem Abschluß wird schließlich neben der Nichtfehler-Meldung 0 in D0 in A0 die Kanalkennzahl (channel ID (identifier)) zurückgegeben. Sie muß vom aufrufenden Programm sorgfältig gespeichert werden, denn unter dieser Kennzahl werden später Ausgaben zu diesem Fenster geleitet. Die Vektorisierung strafft also die Programmgestaltung.

```
0000C8: 003A6A                          = UT SCR

003A6A  LEA     004C(PC).A0     A0=003AB8
003A6E  BSR.S   003A98
  003A98  MOVE.L  A1.-(A7)
  003A9A  MOVEQ   #01.D0        IO OPEN
  003A9C  MOVEQ   #FF.D1        DIES Programm
  003A9E  MOVEQ   #00.D3        schon gelinkt
  003AA0  TRAP    #2            Aufruf
  003AA2  BRA.S   003AA8

  003AA8  MOVEA.L (A7)+.A1
  003AAA  TST.L   D0            Fehlerabfrage
  003AAC  BEQ.S   003AB0
  003AAE  ADDQ.W  #04.A7        abbrechen
  003AB0  RTS
003A70  ADDQ.W  #04.A1
003A72  MOVEQ   #0D.D0          SD WDEF
003A74  MOVEQ   #00.D2          Rahmenbreite
003A76  BSR.S   003AA4
  003AA4  MOVE.L  A1.-(A7)
  003AA6  TRAP    #3            Aufruf
  003AA8  MOVEA.L (A7)+.A1
  003AAA  TST.L   D0            Fehlerabfrage
  003AAC  BEQ.S   003AB0
  003AAE  ADDQ.W  #4.A7         abbrechen
  003AB0  RTS
003A78  SUBQ.W  #04.A1
003A7A  MOVEQ   #0C.D0          SD BORDR
003A7C  MOVE.B  (A1)+.D1        Farbe des Randes
003A7E  MOVE.B  (A1)+.D2        Randbreite
003A80  BSR.S   003AA4
  003AA4  MOVE.L  A1.-(A7)
  003AA6  TRAP    #3            Aufruf
  003AA8  MOVEA.L (A7)+.A1
  003AAA  TST.L   D0            Fehlerabfrage
  003AAC  BEQ.S   003AB0
  003AAE  ADDQ.W  #4.A7         abbrechen
  003AB0  RTS
003A82  MOVEQ   #$27.D0         SD SETPA
003A84  MOVE.B  (A1).D1         Hintergrundfarbe
003A86  BSR.S   003AA4 ....     usw.
```

Bild 32.2 Eine vektorisierte QDOS-Routine

32.4 Ablauf eines TRAP-Befehls und Rückkehr

Ein TRAP-Befehl wird in folgenden Schritten abgearbeitet:

1. Der Prozessor geht in den Supervisor-Modus über; A7 bezeichnet jetzt den SSP, während der Anwender-Stapel nun USP heißt; das S-Bit wird gesetzt.
2. Der Inhalt des Programmzählers wird wie mit −(A7), aber auf dem Supervisor-Stapel abgelegt.
3. Der Inhalt des Statusregisters wird (als Wort) ebenso abgelegt. (Während z.B. bei einem BSR vier Bytes gerettet werden, sind es hier sechs (4 + 2).)
4. Der TRAP-Vektor wird in den Programmzähler geladen.

Aus 3. folgt, daß man nicht einfach mit RTS zurückkehren kann, da dabei nur vier Bytes vom Stapel geholt werden.

Nun schließt sich, wie schon angedeutet, eine so oder so ausgelegte Mehrfachauswahl an, über die die gewünschte Supervisor-Routine erreicht wird. In den User-Modus kann man durch Manipulation des Statusregisters mittels logischer Verknüpfungen zurückgelangen, normalerweise aber wohl mit dem schon in Abschnitt 31.2 erwähnten privilegierten Befehl RTE (return from exception), der nur eingesetzt werden darf, solange sich der M68000 im Supervisor-Modus befindet; anderenfalls erfolgt eine Fehlermeldung «Privilegverletzung». RTE holt das oberste Wort vom Supervisor-Stapel in das Statusregister und das folgende Langwort in den Programmzähler. Hieraus folgt, daß der M68000 in den Modus zurückkehrt, aus dem er in die momentan beendete Exception eintrat.

33
Experiment im Supervisor-Modus

Zur Vorbereitung muß geklärt werden, welche privilegierten Befehle zum Experimentieren einladen und wie man aus dem User- in den Supervisor-Modus gelangt. Als Beispielmaschine dient der diesbezüglich am besten dokumentierte QL.

33.1 Von den privilegierten Befehlen

Versucht man, aus der User-Ebene einen privilegierten Befehl einzusetzen, führt das zur Fehlermeldung «Privilegverletzung». Zu diesen Befehlen gehören:

RTE (Return from exception), wurde bereits besprochen.

MOVE USP,An und MOVE An,USP
Im Multitasking können Zugriffe zum User-Stapel erforderlich werden. Da ein User sich aber selbst einen Stapel einrichten kann, scheint ein Experiment mit diesen Befehlen nicht viel Neues zu bringen.

STOP #Konstante
Die Konstante wird in das Statusregister geladen, und der Prozessor geht in den HALT-Status, d.h., er stellt seine Tätigkeit ein, bis
a) ein Interrupt höherer Priorität (als die Maske) registriert wird,
b) ein RESET durchgeführt wird oder
c) das Trace-Bit gesetzt wurde, mit dem wir uns bald genauer beschäftigen werden.

RESET versetzt alle mit der Reset-Leitung verbundenen peripheren Geräte (z.B. Festplatte, Drucker) in einen definierten Anfangszustand, nicht aber den Prozessor selbst.

ANDI, ORI, EORI, MOVE to SR
Mit diesen Befehlen kann man den Inhalt des Statusregisters verändern, wenn man sich schon im Supervisor-Modus befindet. Wie aber gelangt man in diesen?

33.2 TRAP #0 beim QL

Schaut man im Speicher nach, findet man unter TRAP #0 (vgl. Bild 32.1) folgenden kleinen, aber feinen Zweizeiler:

```
000308  544F  ADDQ.W  #2,A7
00030A  4E75  RTS
```

Offenbar wird das obere Datenwort im Stapel verworfen, worauf es mit RTS zurück zum aufrufenden Programm geht. Das obere Datenwort ist aber der Inhalt des Statusregisters (normalerweise mit S-Bit = 0) vor dem TRAP. Dieses Statuswort wird also ignoriert, und der Prozessor befindet sich bei Rückkehr ins User-Programm im Supervisor-Modus. Hier haben wir den gesuchten Einstieg!

33.3 Wo befindet sich der Supervisor-Stapel?

Die Subroutine LISTAP LIest den STAPel aus und kopiert einige Daten fortlaufend in den Puffer. Als erster wird der momentane Stand des Stapelzeigers dort festgehalten. Dann übernimmt A4 den um 6 verminderten Inhalt des Stackpointers, und mit seiner Hilfe werden 6 Datenwörter aus dem Stapel ausgelesen.

Das Hauptprogramm richtet zuerst A3 auf den Puffer und ruft LISTAP auf. So wird zunächst einmal der User-Stapel dokumentiert. Bevor der weitere Ablauf mit TRAP #0 in den Supervisor-Modus verlegt wird, wird die Rückkehradresse nach A2 und der Inhalt des Statusregisters nach D1 gerettet. Nun archiviert LISTAP die entsprechenden Daten des Supervisor-Stapels.

Ab USERM geht es wieder im User-Modus weiter, und zur Gegenkontrolle wird LISTAP noch einmal aufgerufen. Zur Rückkehr in den normalen Betriebszustand wurde vorher zuerst die gerettete Rückkehradresse auf dem Supervisor-Stapel abgelegt, danach der Inhalt des Statusregisters. Der Rücksprung selbst wird mit RTE erreicht.

```
03F800                          ORG     $3F800
03F800
03F800  47FA 002A               LEA     BUFFER(PC),A3
03F804  6116                    BSR.S   LISTAP
03F806  45FA 000E               LEA     USERM(PC),A2
03F80A  40C1                    MOVE    SR,D1
03F80C  4E40                    TRAP    #0
03F80E  610C                    BSR.S   LISTAP
03F810  2F0A                    MOVE.L  A2,-(A7)
03F812  3F01                    MOVE.W  D1,-(A7)
03F814  4E73                    RTE
03F816
03F816  6104            USERM:  BSR.S   LISTAP
03F818  7000                    MOVEQ   #0,D0
03F81A  4E75                    RTS
03F81C
03F81C  26CF            LISTAP: MOVE.L  A7,(A3)+
03F81E  49EF FFFA               LEA     -6(A7),A4
03F822  7005                    MOVEQ   #5,D0
03F824  36DC            LPSTAP: MOVE.W  (A4)+,(A3)+
03F826  51C8 FFFC               DBRA    D0,LPSTAP
03F82A  4E75                    RTS
03F82C
03F82C                  BUFFER: DS.W    24
03F85C
03F85C                          END
```

P.33.1 Auslesen des Supervisor-Stapels

Das Programm ist zum Ablauf aus dem Monitor (P.0.2) heraus bestimmt, weil so die Ergebnisse am bequemsten ausgelesen werden können. Ebenfalls aus Bequemlichkeitsgründen wurde es auf dem NDR-Computer assembliert (Bild 33.1).

Der Speicherauszug lehrt, daß der User-Stapel dicht vor dem von RESPR reservierten Bereich eingerichtet wurde. Auch auf die zugrundeliegende Speicherdisposition soll noch eingegangen werden.

Der SSP enthielt bei der Momentaufnahme 02847C, das ist etwa 2 KB oberhalb des Beginns der Systemvariablen (und von da bis zum RESPR-

```
03F82C: 03F7C2  03F83C: 02847C  03F84C: 03F7C2
        035C            03E068          035C
        0000                            0000
        0F28            0004            0F28
        03F806          03F810          03F818
        0000            0000            0000
```

Bild 33.1 Auslesen des User-, des Supervisor- und des User-Stapels

Bereich noch einmal rund 90 KB). Sie finden in den drei Stapelauszügen präzise die Rücksprungadressen nach den drei LISTAP-Aufrufen wieder, wohlgemerkt die erste und dritte im User-Stapel, die zweite im Systemstapel.

A.33.1: Über den Stapeleintrag 03E068 soll nicht spekuliert werden. Was aber könnte 0004 im Supervisor-Stapel bedeuten? Experimentieranregung?

33.4 Modus-Experiment mit dem NDR-Computer

Der NDR-Computer befindet sich normalerweise im Supervisor-Modus. Interessenten werden auf Anhang E verwiesen.

34
Die Anmeldung eines Programms beim Betriebssystem QDOS

In Abschnitt 28.7 wurden die Möglichkeiten aufgezählt, ein Maschinenprogramm im QL zum Ablauf zu bringen. Hier soll nun ein Programm korrekt bei QDOS angemeldet werden, so daß es vom Betriebssystem regelrecht betreut wird und ganz offiziell die ihm zukommende Prozessorzeit zugeteilt bekommt, aber auch anderen Jobs die Zuteilung von Rechenzeit nicht verwehrt. Logischerweise müßte ein Programm, das beim Betriebssystem angemeldet wird, zum Schluß auch wieder abgemeldet werden. Wegen der Fülle des zu Beachtenden stellen wir diese Aufgabe zurück und leiten den Ablauf in eine endlose Schleife.

34.1 Projekt «Bremer Stadtmusikanten»

Nach einer norddeutschen Sage wurden ein Hahn, eine Katze, ein Hund und ein Esel wegen Altersschwäche davongejagt. Sie kamen bei Nacht an einer finsteren Herberge an und vertrieben mit ihrem Geschrei die bösen Räuber. Wir wollen die vier Tiere letztlich im Multitasking unabhängig voneinander in verschiedenen Fenstern über den Bildschirm laufen lassen und beginnen – ganz oben – mit dem Hahn.

34.2 Der Programmkopf und die Priorität

P.34.1 wird letztlich mit exec_mdv2_hahn_c gestartet und läuft dann als selbständiges Programm, als eins unter womöglich mehreren. Damit es vom Betriebssystem als solches erkannt und richtig in die Reihe der evtl. laufenden anderen Programme eingereiht wird, muß der Programmkopf bestimmte Angaben enthalten.

```
        =FFFF                   DIES        EQU     -1
        =000B                   MT_PRIOR    EQU     $0B
        =0007                   IO_SSTRG    EQU     7
        =0011                   SD_TAB      EQU     $11
        =00C8                   UT_SCR      EQU     $C8
                                *
0000'    600C                               BRA.S   START
0002'   =0004                              DS.B    4
0006'    4AFB                               DC.W    $4AFB
0008'    0004                               DC.W    4
000A'    4861 686E              NAME        DC.B    'Hahn'

000E'    700B                   START       MOVEQ   #MT_PRIOR,D0
0010'    72FF                               MOVEQ   #DIES,D1
0012'    7401                               MOVEQ   #1,D2
0014'    4E41                               TRAP    #1
                                *
0016'    3478 00C8                          MOVEA.W UT_SCR,A2
001A'    43FA 006A                          LEA     SCR,A1
001E'    4E92                               JSR     (A2)
                                *
0020'    287C 1234 5678         IDENT       MOVEA.L #$12345678,A4
0026'    283C 8765 4321                     MOVE.L  #$87654321,D4
                                *
002C'    47FA 0038                          LEA     BUFFER,A3
0030'    7A1E                   ROTIER      MOVEQ   #$1E,D5
0032'    4246                               CLR.W   D6
0034'    17B3 6001 6000         L2 MOVE.B 1(A3,D6.W),0(A3,D6.W)
003A'    5246                               ADDQ.W  #1,D6
003C'    51CD FFF6                          DBRA    D5,L2
0040'    176B 0001 001F                     MOVE.B  1(A3),$1F(A3)
0046'    6102                               BSR.S   AUS6AB
0048'    60E6                               BRA.S   ROTIER
```

P.34.1 Bremer Stadtmusikanten: hahn_t (I)

Im sechsten Byte (Zählung ab 0) beginnend muß die Kennzahl $4AFB stehen, gefolgt von der .W-Angabe der Länge des Programmnamens und vom Namen selbst. Diese Anordnung («Programmkopf») wird durch den unmotiviert erscheinenden Sprung (über vier reservierte Bytes hinweg) zum

eigentlichen START erreicht. NB: In den ersten 6 Bytes könnte aber auch ein JMP $xxxxxxxx stehen (vgl. NEWJOB in 37.4).

In Verbindung mit dem exec-Befehl genügt der Managertrap MT_PRIOR, um das Betriebssystem zu veranlassen, den Job einzureihen. Die eigentliche Aufgabe von MT_PRIOR ist es, einem (unter womöglich mehreren) laufenden Programm eine bestimmte Priorität (unterschiedlicher Anspruch auf Rechenzeit) zuzuweisen. Dazu muß in D1 die Kennzahl des Programms (Job ID) eingeschrieben sein, dessen Priorität verändert oder bestimmt werden soll. Hier soll das Programm seine eigene Priorität festlegen; dazu muß der Job ID negativ sein.

In D2 wird eine Priorität von 0 bis 127 vorgegeben.

34.3 Priorität und Scheduler (I)

Einem Programm mit hoher Priorität wird viel Rechenzeit zugewiesen, einem mit geringer Priorität wenig. Folgendes ist denkbar: Ein Multiusersystem betreut eine Ladenkasse und berechnet gleichzeitig aufwendige mathematische Funktionen. Da die Kasse meistens geschlossen ist und da, wenn sie geöffnet wird, nur wenig (verglichen mit höheren Besselschen Funktionen) zu rechnen ist, wäre es falsch, ihr eine höhere Priorität zuzuweisen, denn dann würde die Rechenzeit im wesentlichen mit Warten auf eine Benutzung der Kasse vertan.

Unter QDOS erhält der BASIC-Interpreter die Priorität $20. Für den Hahn erscheint uns im Hinblick auf den Weiterausbau des Programms die geringe Priorität 1 als ausreichend.

Die Rechenzeit verteilt eine Scheduler genannte Routine. Man kann sich das (unter Vorbehalt) so vorstellen, daß ein Timer-Baustein alle fünfzigstel Sekunden einen Hardware-Interrupt auslöst (vgl. Abschnitt 31.3), der den Scheduler veranlaßt, mittels der vorgegebenen Prioritäten zu ermitteln, welcher Job als nächster Rechenzeit erhalten muß.

34.4 Die übrigen Programmteile

Nach der Anmeldung beim Betriebssystem wird ein Fenster eingerichtet; wie bekannt, geschieht das mit UT_SCR. Dabei wird an A0 eine Kanalkennzahl (channel ID) übergeben, über die später dieses Fenster angesprochen werden

kann. Da das später zur Ausgabe benutzte IO_SSTRG (send a string of bytes) den channel ID «preserved», also unverändert läßt, braucht sie in diesem Programm nicht weiter sichergestellt zu werden.

Es folgen zwei scheinbar unsinnige Registerbelegungen, deren Sinn und der anderer unnötiger Umständlichkeiten im nächsten Kapitel deutlich werden wird. Das Rotieren des Textes im Puffer braucht nicht näher erläutert zu werden.

Den Pufferinhalt gibt IO_SSTRG (send a string of bytes) aus. Erforderliche Parameter: D2.W: Anzahl der auszugebenden Zeichen; D3.W: Timeout; A0: channel ID (preserved); A1: Pufferbasis (wird bei der Ausgabe fortgeschrieben). SD_TAB setzt die nächste Schreibposition auf den Zeilenanfang.

```
004A'  7007                AUSGAB   MOVEQ     #IO_SSTRG,D0
004C'  7420                         MOVEQ     #$20,D2
004E'  7602                         MOVEQ     #2,D3
0050'  224B                         MOVEA.L   A3,A1
0052'  4E43                         TRAP      #3
0054'  6108                         BSR.S     WARTEN
0056'  7011                         MOVEQ     #SD_TAB,D0
0058'  7200                         MOVEQ     #0,D1
005A'  4E43                         TRAP      #3
005C'  4E75                         RTS
                           *
005E'  7E00                WARTEN   MOVEQ     #0,D7
0060'  5347                LWART    SUBQ.W    #1,D7
0062'  66FC                         BNE.S     LWART
0064'  4E75                         RTS
                           *
0066'  2020 2020 2020     BUFFER   DC.B      '   '
006C'  2020 2020 2020              DC.B      '   '
0072'  2020 2020 2020              DC.B      '   '
0078'  2020 2020 2020              DC.B      '   '
007E'  2020 2020                   DC.B      '   '
0082'  4861 686E                   DC.B      'Hahn'
0086'  0000 0400           SCR      DC.L      $00000400
008A'  01AB 000B                   DC.L      $01AB000B
008E'  0028 000A                   DC.L      $0028000A
                           *
                                   END
```

P.34.1 Bremer Stadtmusikanten: hahn_t (II)

35
Multitasking

Vor etwa acht Jahren ließ der Verfasser auf dem Urvater PET 2001 Autos nach Maßgabe von Zufallszahlen um die Wette fahren. Im folgenden geschieht auf den ersten Blick nichts anderes, doch Hahn, Katze, Hund und Esel laufen – von je einem eigenen Programm bewegt – unabhängig voneinander in ihren Bahnen. Man kann die vier Programme in beliebiger Reihenfolge und in beliebigen zeitlichen Abständen laden, ja sogar noch andere und ein BASIC-Programm dazu – im Extremfall 56 verschiedene Programme!

35.1 Die drei anderen Tiere

Die Erstellung der drei anderen Programme bereitet kaum Mühe. Man braucht den Editor nur einmal aufzurufen und hahn_t zu laden, dann ändert man an vier Stellen ein wenig ab und savet katz_t, hund_t und esel_t fast in einem einzigen Arbeitsgang. Zu ändern ist:

000A'	Hahn	in	Katz	bzw.	Hund	bzw.	Esel
0012'	1	in	2	bzw.	4	bzw.	8
0082'	Hahn	in	Katz	bzw.	Hund	bzw.	Esel
008E'	0A	in	1A	bzw.	2A	bzw.	3A

Anschließend kann man ganz ähnlich rationell assemblieren.

Sind die Programme dann schließlich mit exec gestartet, will es scheinen, daß sie den Prioritäten entsprechend mit einfacher, doppelter, vierfacher usw. Geschwindigkeit laufen. Dann aber fällt auf, daß Hahn und Katz annäherungsweise gleich schnell sind. Möglicherweise beruht dieser überraschende Effekt darauf, daß ja auch der Scheduler Zeit benötigt und sich störend synchron einschaltet. – Hier ist reichlich Gelegenheit zum Experimentieren. Sie werden schon bald erfahren, wie man die Prioritäten sogar während des Programmablaufs verändern kann.

35.2 Der Job-Kontrollblock

Möglicherweise haben Sie sich schon gefragt, wie es der Prozessor mit «nur» 16 Registern fertigbringt, fünf (oder mehr) verschiedene Programme zu verwalten. Hier kommt die Antwort:

Wir haben schon erwähnt, daß ab $28000 die Systemvariablen in den Speicher eingeschrieben sind. Mit Offset $68, also ab $28068, ist dort SV_JBBAS zu finden (pointer to base of Job table), also die Adresse der Jobtafel, $28A80 (Bild 35.1).

Mit etwas Fingerspitzengefühl kann man aus dem Monitorprogramm P.0.2 einige nicht benötigte und evtl. sogar störende Zeilen entfernen und, wenn man über einen Printer verfügt, einen Speicherauszug drucken, *während* die Bremer Stadtmusikanten laufen. Das «während» ist wichtig, denn man kann nicht erkennen, wann welcher Job aktiviert wurde. Grundsätzlich ist es sogar möglich, daß z.B. einige Datenwörter falsch ausgelesen wurden, weil nach dem Lesen des MSB das BASIC-Programm unterbrochen und ein anderer Job aktiviert wurde, so daß bei Zuteilung neuer Rechenzeit ein anderes Datenwort in der adressierten Speicherstelle steht und das gedruckte MSB und das LSB nicht zusammengehören. Das ist möglich, doch wird zu erkennen sein, daß die Wahrscheinlichkeit dafür nicht sonderlich groß ist.

In der Jobtafel sind fünf Jobs eingetragen; ihre Job-Kontrollblocks beginnen in

03DC00:	BASIC (BASIC-Programme stehen immer vorn)
03FD10:	Esel, als erstes gestartet, Priorität 8;
03FA20:	Katz, als zweites gestartet, Priorität 2;
03F730:	Hahn (?), als drittes gestartet, Pr. = 1;
03F440:	Hund (?), als viertes gestartet, Pr. = 4.

Offenbar sind die Programme nicht in der Reihenfolge ihrer Prioritäten eingetragen, sondern in der Reihenfolge ihres Linkens.

Wir haben nur zwei Job-Kontrollblocks abgedruckt; jeder besteht aus drei Teilen, dem eigentlichen Kontrollblock, dem Maschinenprogramm und dem Datenbereich (Data area).

Zum Objektcode ist nichts zu sagen, außer daß man hier sieht, unter welcher absoluten Adresse er in diesem konkreten Beispiel zum Ablauf gebracht wurde. Der Datenbereich enthält hier im wesentlichen den Stapel, für den automatisch vom Betriebssystem Platz zugewiesen wird: Für jedes Programm wird ein eigener Stapel eingerichtet. Der Datenbereich scheint immer mit dem Langwort 00000000 zu enden.

```
Job table
028A80   0003 DC00 0003 FD10 0003 FA20 0003 F730
028A90   0003 F440 FF00 0000 FF00 0000 FF00 0000

Job control block Katz
03FA20   0000 02F0 0003 FA88 0000 0000 0000 0000
03FA30   0002 2302 0000 0000 0000 0000 0000 0000
03FA40   0000 0000 0000 0020 0000 0020 0000 0002
03FA50   8765 4321 0000 FFFF 0000 001F 0000 CC2E
03FA60   002D 0004 0003 FB0E 0000 3A6A 0003 FAEE
03FA70   1234 5678 0000 0288 0003 FA88 0003 FD04
03FA80   0000 0003 FAE8 0000

03FA88   600C 0000 0000 4AFB 0004 4B61 747A 700B
03FA98   72FF 7402 4E41 3478 00C8 43FA 006A 4E92
03FAA8   287C 1234 5678 283C 8765 4321 47FA 0038
03FAB8   7A1E 4246 17B3 6001 6000 5346 51CD FFF6
03FAC8   176B 0001 001F 6102 60E6 7007 7420 7602
03FAD8   224B 4E43 6108 7011 7200 4E43 4E75 7E00
03FAE8   5347 66FC 4E75 2020 2020 2020 2020 2020
03FAF8   2020 2020 2020 2020 2020 2020 204B 6174
03FB08   7A20 2020 2020 0000 0400 01AB 000B 0028
03FB18   001A

03FB1A   0000 0000 .... ....
03FD00   0003 FB11 0003 FADE 0003 FAD0 0000 0000

Job control block Esel
03FD10   0000 02F0 0003 FD78 0000 0000 0000 0000
03FD20   0001 2908 0000 0000 0000 0000 0000 0000
03FD30   0000 0000 0000 0020 0000 0020 0000 0002
03FD40   8765 4321 0000 FFFF 0000 001F 0000 921D
03FD50   002B 0003 0003 FDFE 0000 3A6A 0003 FDDE
03FD60   1234 5678 0000 0288 0003 FD78 0003 FFF4
03FD70   0008 0003 FDD8 0000

03FD78   600C 0000 0000 4AFB 0004 4573 656C 700B
03FD88   72FF 7408 4E41 3478 00C8 43FA 006A 4E92
03FD98   287C 1234 5678 283C 8765 4321 47FA 0038
03FDA8   7A1E 4246 17B3 6001 6000 5246 51CD FFF6
03FDB8   176B 0001 001F 6102 60E6 7007 7420 7602
03FDC8   224B 4E43 6108 7001 7200 4E43 4E75 7E00
03FDD8   5347 66FC 4E75 2020 7365 6C20 2020 2020
03FDE8   2020 2020 2020 2020 2020 2020 2020 2020
03FDF8   2020 2045 7365 0000 0400 01AB 000B 0028
03FE08   003A

03FE0A   0000 0000 .... ....
03FFF0   0003 FE01 0003 FDCE 0003 FDC0 0000 0000
```

Bild 35.1 Bremer Stadtmusikanten: Speicherauszug

A.35.1: Deuten Sie die Stapeleinträge.

Der Job-Kontrollblock enthält nach DICKENS (s. Literaturverzeichnis) folgende Angaben:

$00.L Gesamtlänge von Job-Kontrollblock und Datenbereich;
$04.L Startadresse bei Aktivierung;
$08.L Job ID eines evtl. übergeordneten (z.B. Host-)Programms;
.....
$12.B Momentan akkumulierte Prioritätssumme;
$13.B Prioritätsinkrement; von BASIC her gestartete Programme erhalten Priorität $20;
$14.W Job-Status: 0000 = nicht suspendiert;
.....
$1C.L Pointer zu umlenkbaren TRAP-Vektoren;
$20 8mal .L: zwischengespeicherte Datenregister;
$40 8mal .L: zwischengespeicherte Adreßregister;
$60.W zwischengespeicherter Inhalt des Statusregisters;
$62.L zwischengespeicherter Programmzählerinhalt.

Wir listen diese Registerinhalte ein bißchen übersichtlicher auf (Bild 35.2).

A.35.2: Deuten Sie unter dem Vorbehalt der zeitlichen Homogenität die Inhalte aller Register.

35.3 Priorität und Scheduler (II)

Die Bytes $12 und $13 des Job-Kontrollblocks enthalten nach DICKENS eine Prioritätssumme und das Prioritätsinkrement. Beim Programm Katz findet sich hier 23 02, bei Esel 29 08. Diese Programme laufen hier mit den Prioritäten 02 bzw. 08; damit dürfte die Bedeutung von Byte $13 hinreichend geklärt sein. Mit der gebotenen Vorsicht darf vermutet werden, daß diese Inkremente in $12 aufaddiert werden und daß bei der Aktivierung des Schedulers der Job mit der höchsten Prioritätssumme gesucht wird. Logischerweise wird dann die Summe erheblich vermindert werden, und der herausgefundene Job erhält Rechenzeit. Im einzelnen könnte das so vor sich gehen:

1. Auftreten des Ebene-2-Interrupts
2. Suspendieren des bisher laufenden Jobs

	Katz	Esel
D0	00000000	00000000
D1	00000020	00000020
D2	00000020	00000020
D3	00000002	00000002
D4	87654321	87654321
D5	0000FFFF	0000FFFF
D6	0000001F	0000001F
D7	0000CC2E	0000921D
A0	002D0004	002B0003
A1	0003FB0E	0003FDFE
A2	00003A6A	00003A6A
A3	0003FAEE	0003FDDE
A4	12345678	12345678
A5	00000288	00000288
A6	0003FA88	0003FD78
A7	0003FD04	0003FFF4
SR	0000	0008
PC	0003FAE8	0003FDD8

Bild 35.2 Die zwischengespeicherten Registerinhalte

3. Retten seiner Registerinhalte usw. in seinen Job-Kontrollblock
4. Starten der Scheduler-Routine
5. Herausfinden des nächsten Rechenzeitempfängers
6. Suspendieren des Schedulers
7. Laden der Register usw. aus dem Kontrollblock des neuen Jobs
8. Reaktivieren des neuen Jobs an genau der Stelle, an der er suspendiert wurde.

Der Status eines Jobs innerhalb eines Multiuser-Systems kann sein:

a) aktiv (das Programm läuft gerade);
b) suspendiert (die Priorität ist >0, doch das Programm wartet auf die nächste Rechenzeitzuteilung);
c) inaktiv (ein solcher Job schläft im Speicher, bis seine Priorität verändert oder er selbst aus dem Speicher [transient program area] entfernt wird).

A.35.3: Wie kann man die Priorität der einzelnen Stadtmusikanten sehr einfach verändern?

36
Tracing

Unter Tracing versteht man das schrittweise Abarbeiten eines Programms (single step mode) zum Zwecke der Fehlersuche. In Abschnitt 31.1 wurde das T-Bit des SR erwähnt; hier soll an einem Beispiel gezeigt werden, was es damit auf sich hat. Voraussetzung ist ein gut dokumentierter Computer mit entsprechendem Betriebssystem; der QL erfüllt diese Voraussetzung recht gut. (Das Grundprogramm des NDR enthält eine residente Einzelschritt-Option.)

Die QL-Besitzer werden um Nachsicht gebeten; der Rechner des Verfassers (QDOS V1.03) bleibt im Programmschluß hängen. Über mögliche Ursachen soll nachgedacht werden. Weil aber das Programm das Tracing als Feature des M68000 einwandfrei vorführt und Feinheiten und Besonderheiten von QDOS nicht Gegenstand dieses Buches sind, wurde P.36.1 trotz des Mangels aufgenommen.

36.1 Zum Programmentwurf

An P.36.1 sollen wieder verschiedene Dinge gezeigt werden:

1. Wie gelangt man in den Trace-Modus? Wie etwa ist eine Trace-Routine aufgebaut? Wie gelangt man wieder heraus?
2. Das rechenintensive (im Gegensatz zu I/O-intensiv) Kurzprogramm berechnet die Quadratwurzel.
3. Wie kann man eine Eingabe in ein selbständiges Maschinenprogramm tätigen?
4. Wie wird ein selbständiges Programm, nachdem es seinen Zweck erfüllt hat, aus dem Speicher entfernt?

Das Programm ist ziemlich konsequent modular aufgebaut; so läßt sich schnell übersehen, was geschieht: Auf den in 34.2 besprochenen Programmkopf folgen die Moduln

```
                =FFFF                   DIES        EQU         -1
                =000B                   MT_PRIOR    EQU         $0B
                =0005                   MT_FRJOB    EQU         5
                =0007                   MT_TRAPV    EQU         7
                =0002                   IO_CLOSE    EQU         2
                =0001                   IO_FBYTE    EQU         1
                =0002                   IO_FLINE    EQU         2
                =0007                   IO_SSTRG    EQU         7
                =0012                   SD_NL       EQU         $12
                =0018                   SD_SCROL    EQU         $18
                =0011                   SD_TAB      EQU         $11
                =00C6                   UT_CON      EQU         $C6
                =00C8                   UT_SCR      EQU         $C8
                =00FE                   CN_ITOHL    EQU         $FE
0000'            600E                               BRA.S       START0
0002'           =0004                               DS.B        4
0006'            4AFB                               DC.W        $4AFB
0008'            0006                               DC.W        6
000A'            5452 4143 4520                     DC.B        'TRACE
0010'            700B                   START0      MOVEQ       #MT_PRIOR,D0
0012'            72FF                               MOVEQ       #DIES,D1
0014'            7401                               MOVEQ       #1,D2
0016'            4E41                               TRAP        #1

0018'            6100 001A              START1      BSR         SCREEN
001C'            6100 0030                          BSR         CONSOL
0020'            6100 0046                          BSR         TRAVEK
0024'            6100 00B4                          BSR         EINLES
0028'            6100 009A                          BSR         TRACON
002C'            6100 00E8                          BSR         WURZEL
0030'            6000 0168                          BRA         ABBAU

0034'            347B 00C8              SCREEN      MOVEA.W     UT_SCR,A2
0038'            43FA 0008                          LEA         SCR,A1
003C'            4E92                               JSR         (A2)
003E'            2848                               MOVEA.L     A0,A4
0040'            4E75                               RTS
0042'            0000 0400              SCR         DC.L        $00000400
0046'            01CB 0080                          DC.L        $01CB0080
004A'            0008 000A                          DC.L        $0008000A
```

P.36.1 Tracing (I)

SCREEN: Aufmachen eines Ausgabefensters wie in Abschnitt 32.3 besprochen;
CONSOL: Eröffnen eines Kanals zur Tastatureingabe mit Echo auf dem Bildschirm, ganz analog zu UT_SCR;
TRAVEK: Hier können (denken Sie z.B. an den Job-Kontrollblock) userspezifische Ausnahme/Exception-Vektoren definiert werden, darunter ein Trace-Vektor;
EINLES: Eingabe des Radikanden nach D4;
TRACON: Einschalten des Trace-Modus;
WURZEL: Radiziert wird im Single-step-Modus mit Anzeige von vier Registerinhalten;
ABBAU: Nach Durchlauf soll (s.o.) das Programm inaktiviert und aus dem Speicher entfernt werden.

Ausgabe- und I/O-Kanal werden mittels vektorisierter Routinen eröffnet. Die von QDOS in A0 zurückgegebenen Kanalkennzahlen werden in A4 (SCREEN) und in A5 (CONSOL) bleibend zwischengespeichert.

36.2 Vorbereitung und Einschalten des Trace-Modus

Mit dem Manager-Trap MT_TRAPV, der nicht mit dem fast gleichlautenden Maschinenbefehl TRAPV verwechselt werden darf, kann jeder User 19 eigene Trap- und andere Ausnahmevektoren definieren. Sie werden in einer VEKtorTABelle bereitgestellt und mittels MT_TRAPV in das Betriebssystem einbezogen.

Um nicht weit übers Thema hinauszuschießen, aber auch mit einigem Leichtsinn wurde von diesen 19 Vektoren nur einer wirklich definiert. Zur Vorführung des Tracing genügt das; wehe nur, wenn ein anderer Ausnahmezustand eintritt! Wie die Speichereinträge zeigen, wurde eine relative Adresse eingetragen, was erkennen läßt, daß hier ein Register (es ist, wie schon erwähnt, A6) eine Basisadresse enthält; der Vektoreintrag wird offenbar als Index draufgesetzt.

D0 = −1 legt ausdrücklich fest, daß das Programm TRACE diese Vektoren für sich selbst (DIES) und nicht etwa für ein anderes Programm festlegt.

Zum Einschalten des Trace-Modus muß das T-Bit des SR gesetzt werden, was so oder so nur im Supervisor-Modus geschehen kann; des modularen Aufbaus willen soll das in einer Subroutine geschehen. TRAP #0 versetzt den

```
004E'   3478 00C6        CONSOL   MOVEA.W   UT_CON,A2
0052'   43FA 0008                 LEA       CON,A1
0056'   4E92                       JSR       (A2)
0058'   2A48                      MOVEA.L   A0,A5
005A'   4E75                       RTS
005C'   0000 0400        CON      DC.L      $00000400
0060'   01CB 0014                 DC.L      $01CB0014
0064'   0008 0094                 DC.L      $00080094

0068'   7007             TRAVEK   MOVEQ     #MT_TRAPV,D0
006A'   72FF                      MOVEQ     #DIES,D1
006C'   43FA 0006                 LEA       VEKTAB,A1

0070'   4E41                       TRAP      #1
0072'   4E75                       RTS
0074'  =0018             VEKTAB   DS.L      6
008C'   0000 012C                 DC.L      AUSGEB
0090'  =0034                       DS.L      13

00C4'   41FA 0012        TRACON   LEA       EXTRON,A0
00C8'   4E40                       TRAP      #0
00CA'   4FEF FFFA                  LEA       -6(A7),A7
00CE'   3EBC 8000                  MOVE.W    #$8000,(A7)
00D2'   2F48 0002                  MOVE.L    A0,2(A7)
00D6'   4E73                       RTE
00D8'   4E75             EXTRON   RTS

00DA'   2F0E             EINLES   MOVE.L    A6,-(A7)
00DC'   9DCE                       SUBA.L    A6,A6
00DE'   7002                       MOVEQ     #IO_FLINE,D0
00E0'   740A                       MOVEQ     #10,D2
00E2'   363C 05DC                  MOVE.W    #1500,D3
00E6'   204D                       MOVEA.L   A5,A0
00E8'   43FA 00D8                  LEA       BUFFER,A1
00EC'   4E43                       TRAP      #3
00EE'   2C5F                       MOVEA.L   (A7)+,A6
```

P.36.1 Tracing (II)

Prozessor in den S-Mode (vgl. Abschnitt 33.2). A7 wird um 6 vermindert und zeigt wieder auf den im Supervisorstapel abgelegten Inhalt des Statusregisters. Hier erhält dieser Stapeleintrag einen anderen Wert (vgl. auch Kommentar zum STOP-Befehl in einem M68000-Handbuch). Als Rücksprungadresse wird die vom RTS bei EXTRON eingetragen. Das RTE läßt nun den Programmablauf dort mit gesetztem T-Bit fortsetzen.

36.3 Von der Console ins Register

IO_FLINE liest eine mit cr abgeschlossene Zeile mit Echo von der Console ein; der eingegebene String steht dann im Puffer, auf den A1 zeigt. IO_FLINE (fetch line) erwartet hier eine Adreßdistanz ab Programmbeginn, auf den ja (siehe Besprechung des Job-Kontrollblocks) A6 zeigt; wir haben hier den Weg gewählt, A6 vorübergehend auf Null zu setzen; denkbar wäre auch SUBA.L A6,A1. Die Zeitbegrenzung auf 1500 Wechselstromzyklen (30 Sekunden) für die ganze Eingabe müßte ausreichen. Das Consol-Fenster muß vorher durch CTRL-C aktiviert werden.

Die Eingabe steht nun als String im Puffer, doch leider funktionieren die CoNvertierungsroutinen CN_HTOIx unter QDOS V1.03 offensichtlich (und nach DICKENS) nicht; deshalb ermittelt eine eigene Routine den Zahlenwert und lädt ihn nach D4.

36.4 Wurzelziehen durch Bit-KO

Ein KO-Algorithmus klärt systematisch eine Alternative nach der anderen. Eine Alternative in diesem Sinne ist eine Frage, die sich entweder mit Ja oder mit Nein beantworten läßt. Eine Alternativfrage wählt eine von zwei Möglichkeiten, zwei identifizieren eine aus vier, drei eine aus acht usw. Man kann auch nach KO-System sortieren: Gehört das neue Element zur ersten oder zur zweiten Hälfte der eingespeicherten Daten? Auch beim heiteren Beruferaten sind neben den Imponderabilien direkter persönlicher Beobachtung Alternativstrukturen erkennbar (Sind Sie mit der Herstellung oder Verteilung einer Ware beschäftigt?).

Im folgenden wird in einem Ganzzahlrechenwerk die Wurzel aus einer .L-Zahl gezogen, genauer: Es wird die Zahl ermittelt, deren Quadrat kleiner oder gleich dem Radikanden ist. Der Reihe nach wird geprüft, ob ein Bit des

```
00F0'  7800           STR8HX   MOVEQ    #0,D4
00F2'  43FA 00CE               LEA      BUFFER,A1
00F6'  6100 0002               BSR      EINLS2
00FA'  6100 0002      EINLS2   BSR      EINLS3
00FE'  6100 0002      EINLS3   BSR      EINLS4
0102'  E984           EINLS4   ASL.L    #4,D4
0104'  1A19                    MOVE.B   (A1)+,D5
0106'  0C05 003A               CMPI.B   #'3A,D5
010A'  6502                    BCS.S    NOTSUB
010C'  5F05                    SUBQ.B   #7,D5
010E'  0205 000F      NOTSUB   ANDI.B   #'F,D5
0112'  D805                    ADD.B    D5,D4
0114'  4E75                    RTS

0116'  7A0F           WURZEL   MOVEQ    #15,D5
0118'  7C00                    MOVEQ    #0,D6
011A'  0BC6           LOOP     BSET     D5,D6
011C'  3E06                    MOVE.W   D6,D7
011E'  CEC7                    MULU     D7,D7
0120'  B887                    CMP.L    D7,D4
0122'  6402                    BCC.S    LAB1
0124'  0B86                    BCLR     D5,D6
0126'  51CD FFF2      LAB1     DBRA     D5,LOOP
012A'  4E75                    RTS

012C'  6100 0038      AUSGEB   BSR      HEXASC
0130'  7007                    MOVEQ #IO_SSTRG,D0
0132'  7423                    MOVEQ    #35,D2
0134'  76FF                    MOVEQ    #-1,D3
0136'  204C                    MOVEA.L  A4,A0
0138'  43FA 0088               LEA      BUFFER,A1
013C'  4E43                    TRAP     #3

013E'  7001           KEYIN    MOVEQ #IO_FBYTE,D0
0140'  7605                    MOVEQ    #5,D3
0142'  204D                    MOVEA.L  A5,A0
0144'  4E43                    TRAP     #3
0146'  4A80                    TST.L    D0
0148'  66F4                    BNE.S    KEYIN
```

P.36.1 Tracing (III)

Wurzelwerts gesetzt oder gelöscht sein muß. Der Ansatz ist ausbaufähig: Man könnte die Stellengenauigkeit mit wenig Aufwand verdoppeln und sogar in einem Festkomma-Rechenwerk Nachkommastellen berechnen. Geht das nicht auch in einem FP-Rechenwerk?

Sechzehn Bits sind zu prüfen; dementsprechend wird der Schleifenzähler D5 mit 15 geladen. D5 ist zugleich Zeiger auf das geprüfte Bit des in D6 aufzubauenden Wurzelwerts. Die dekrementative Struktur der DBRA-Schleife kommt unserem Vorhaben entgegen. Das mit dem in D4 stehenden Radikanden zu vergleichende Momentanprodukt wird in D7 erzeugt.

Die Wurzelberechnung ist im modularen Programm als Subroutine ausgelegt, nach deren Abarbeitung das Programm abgebaut, aus dem Speicher entfernt wird. Eine Anzeige war bisher nicht zu erkennen.

36.5 Die Anzeige im Trace-Modus

Zum Entwurf einer T-Anzeigeroutine muß man wissen, daß die bisher besprochenen Programmteile mit SR = 80xx durchlaufen wurden; im Systembyte war nur das Trace-Bit gesetzt. Dieses veranlaßt den Prozessor, nach der Ausführung eines jeden Befehls eine durch Exception-Vektor definierte Routine im Supervisor-Modus mit SR = 2000 zu durchlaufen, in unserem Beispiel ist das AUSGEB.

Ausgegeben werden die Inhalte der vier am Radizieren beteiligten Register D4 ... D7. Die vier achtstelligen Anzeigen sind durch je ein Space getrennt, was eine 35stellige Anzeige ergibt, die vom HEXASC aufgebaut und vom schon bekannten IO_SSTRG (vgl. Abschnitt 34.4) auf den Bildschirm gebracht wird. KEYIN hält das Programm mittels IO_FBYTE (fetch a byte) bis zum nächsten Tastendruck auf der Stelle fest. Zeilenvorschub und Scrollen sind von Abschnitt 19.3 her bekannt.

```
014A'  7012      MOVEQ    #SD_NL,D0
014C'  76FF      MOVEQ    #-1,D3
014E'  204C      MOVE.L   A4,A0
0150'  4E43      TRAP     #3
0152'  4A80      TST.L    D0
0154'  670E      BEQ.S    AUSEXI
0156'  7018      MOVEQ    #SD_SCROL,D0
0158'  72F6      MOVEQ    #-10,D1
```

```
015A'  76FF                    MOVEQ    #-1,D3
015C'  4E43                    TRAP     #3
015E'  7011                    MOVEQ    #SD_TAB,D0
0160'  7200                    MOVEQ    #0,D1
0162'  4E43                    TRAP     #3
0164'  4E73            AUSEXI  RTE

0166'  41FA 005A       HEXASC  LEA      BUFFER,A0
016A'  2204                    MOVE.L   D4,D1
016C'  6100 0010               BSR      UMWAND
0170'  2205                    MOVE.L   D5,D1
0172'  6100 000A               BSR      UMWAND
0176'  2206                    MOVE.L   D6,D1
0178'  6100 0004               BSR      UMWAND
017C'  2207                    MOVE.L   D7,D1
017E'  2F0E            UMWAND  MOVE.L   A6,-(A7)
0180'  9DCE                    SUBA.L   A6,A6
0182'  43FA 0012               LEA      HXZAHL,A1
0186'  2281                    MOVE.L   D1,(A1)
0188'  3478 00FE               MOVEA.W  CN_ITOHL,A2
018C'  4E92                    JSR      (A2)
018E'  2C5F                    MOVEA.L  (A7)+,A6
0190'  10FC 0020               MOVE.B   #' ',(A0)+
0194'  4E75                    RTS
0196'  0000 0001       HXZAHL  DC.L     1
```

P.36.1 Tracing (IV)

Zu HEXASC ist nur noch zu vermerken, daß CN_ITOHL den Pufferzeiger A0 fortschreibt, so daß nach jeder .L-Umwandlung unmittelbar ein Space angefügt werden kann.

A.36.1: Weshalb endet AUSGEB mit RTE statt RTS?

36.6 Die Selbstentfernung des Programms

Nach Ablauf des eigentlichen Wurzelprogramms wird ganz analog zu TRACON der Trace-Modus ausgeschaltet. IO_CLOSE schließt dann die Kanäle mit den in A4 und A5 zwischengespeicherten Kennzahlen, nämlich SCREEN und CONSOL. Wenn diese Kanäle nicht mehr verfügbar sind, kann DIESes

Programm auch keine Tastatureingaben mehr registrieren, und der Computer bleibt ohne Kontakt zu seiner Umwelt. Es wäre aber zu einfach, den eingangs erwähnten Hänger hiermit zu erklären. Wir sollten nicht vergessen, daß von den zwanzig möglichen Vektoren in VEKTAB nur einer definiert wurde, während die anderen wirre Zufallszahlen enthalten.

Die Inaktivierung DIESes Programms und seine Entfernung aus dem Speicher erledigt der Manager-TRAP Force Remove JOB. Aus dem Speicher können nur inaktive Jobs entfernt werden; inaktive Jobs können aber gar nichts tun, auch nicht sich selbst entfernen. MT_FRJOB kann dennoch beides. Nun kann man aber an einem blockierten Rechner nicht kontrollieren, ob ein Programm tatsächlich entfernt wurde. QL-Besitzer, die dem nachgehen wollen, brauchen nur TRACON auszulassen und TRACOF (bei ABBAU) zu unterdrücken. Eine Auflistung der Jobtafel zeigt zwar noch an, wo der Job-Kontrollblock begann, doch das MSB des Eintrags ist FF: Der Job existiert nicht mehr.

```
019A' 41FA 0012    ABBAU   LEA       CTRLC,A0
019E' 4E40         TRACOF  TRAP      #0
01A0' 4FEF FFFA            LEA       -6(A7),A7
01A4' 3EBC 0000            MOVE.W    #0,(A7)
01A8' 2F48 0002            MOVE.L    A0,02(A7)
01AC' 4E73                 RTE
01AE' 7002         CTRLC   MOVEQ     #IO_CLOSE,D0
01B0' 204C                 MOVEA.L   A4,A0
01B2' 4E42                 TRAP      #2
01B4' 7002                 MOVEQ     #IO_CLOSE,D0
01B6' 204D                 MOVEA.L   A5,A0
01B8' 4E42                 TRAP      #2
01BA' 7005                 MOVEQ     #MT_FRJOB,D0
01BC' 72FF                 MOVEQ     #DIES,D1
01BE' 7600                 MOVEQ     #0,D3
01C0' 4E41                 TRAP      #1

01C2' =0014        BUFFER  DS.B      20

                           END
```

P.36.1 Tracing (V)

37
Another self-cloning program

Die etwas verblüffende Kapitelüberschrift ist eine Reverenz an Mr. Adrian DICKENS und soll die geistige Wurzel aufdecken: Das unübertrefflich schön formulierte «self-cloning program» von A. DICKENS finden Sie auf S. 55 bis 59 seines Buches (s. Anhang A und Kapitel 38). Es geht dort wie hier um ein Programm, das neue und im Prinzip selbständige Programme erzeugt. Diese sollen sich als solche auf dem Bildschirm zu erkennen geben.

37.1 Programmentwurf

Um die doch recht komplexen Zusammenspiele beim Cloning zu verdeutlichen, entwerfen wir ein Masterprogramm MCLONE, das unter dem Namen «ABCusw» beim Betriebssystem angemeldet wird und nichts anderes tut, als Tochter- (oder Slave-)Programme hervorzubringen und unter dem Namen «Sabcus» in den Gesamtablauf einzubringen. Diese Tochter- (oder auch F-) Programme geben lediglich ihre Kennbuchstaben (bis zu) neunmal auf dem gemeinsamen SCREEN aus und inaktivieren sich dann, indem sie ihre eigene Priorität auf Null setzen, was bei FERTIG geschieht. Auch MCLONE setzt sich auf diesem Wege zur Ruhe, nachdem es seine Aufgabe erfüllt hat.

Wir besprechen als erstes das letzte, das was von diesem Programmhaufen nach Ablauf im Speicher übrigbleibt. Bild 37.1 wurde so ermittelt: Zuerst lädt man das Monitorprogramm, entfernt alles, was nicht zum Hexdump benötigt wird, insbesondere die Speicherraumreservierung, und ergänzt die Druckanweisung. So wird sichergestellt, daß nach Ablauf des sich selbst vermehrenden Programms in der Jobtafel *und* im Speicher noch Platz für das BASIC-Programm ist. Dann wird das SC-Programm mit exec gestartet. Erst wenn es abgelaufen ist, startet man mit RUN das BASIC-Programm.

Bild 37.1 beginnt mit der – hier verkürzten – Jobtafel, die Platz für insgesamt 56 Eintragungen bietet: Der erste weist zum Job-Kontrollblock des

```
Job table
028A80  0003C800 0003FCD0 0003FC50 0003FBD0
028A90  0003FB50 0003FAD0 0003FA50 ........
028B50  0003E350 0003E2D0 0003E250 0003E1D0

Job control block F-2 (gibt jetzt 'D' aus)
03FBD0  0000 0080 0003 FC38 0001 0002 0000 0000
03FBE0  0002 0000 0000 0000 0000 0000 0000 0000
03FBF0  0000 0000 0002 0003 0000 0000 0000 0008
03FC00  0000 FFFF 0000 0044 000B 000C FFFF FFFF
03FC10  0003 FBD0 0000 0000 0000 0000 0000 0000
03FC20  0006 0003 0000 0018 0003 F7D0 0003 FC50
03FC30  0004 0003 FD8A 0000

03FC38  4EF9 0003 FD5A 4AFB 0006 5361 6263 7573

03FC48  0000 0000 0003 FD76

Job control block F-1 (gibt jetzt 'D' aus)
03FC50  0000 0080 0003 FCB8 0000 0001 0000 0000
03FC60  0001 0000 0000 0000 0000 0000 0000 0000
03FC70  0000 0000 0001 0002 0000 0000 0000 0008
03FC80  0000 FFFF 0000 0044 0026 0027 FFFF FFFF
03FC90  0003 FC50 0000 0000 0000 0000 0000 0000
03FCA0  0006 0003 0000 0018 0003 EA50 0003 FCD0
03FCB0  0004 0003 FD8A 0000

03FCB8  4EF9 0003 FD5A 4AFB 0006 5361 6263 7573

03FCC8  0000 84E7 0003 FD76

Job control block Master
03FCD0  0000 0330 0003 FD38 0000 0000 0000 0000
03FCE0  0000 0000 0000 0000 0000 0000 0000 0000
03FCF0  0000 0000 0000 0001 0000 0000 0000 0000
03FD00  0000 0000 0000 0041 0001 0002 0000 0000
03FD10  0003 FCD0 0003 FD5A 0000 3A63 0000 0000
03FD20  0006 0003 0000 02BC 0003 FCD0 0003 FFF0
03FD30  0004 0003 FD8A 0000

03FD38  600E 0000 0000 4AFB 0006 4142 4375 7377
03FD48  6100 009E 2848 7A40 6100 0038 2C01 6000
03FD58  002A 7005 1205 76FF 204C 4E43 4A80 661A
03FD68  7008 72FF 760A 93C9 4E41 6100 0016 7EFF
03FD78  4E71 51CF FFFC 51CC FFDA 700B 72FF 7400
03FD88  4E41 7001 72FF 7410 7608 93C9 4E41 4A80
03FD98  6702 4E75 2C48 2C01 3CFC 4EF9 43FA FFB4
03FDA8  2CC9 3CFC 4AFB 2CFC 0006 5361 2CBC 6263
03FDB8  7573 4DE8 0018 700B 7400 4E41 217C 0000
03FDC8  0008 0030 5385 2145 0034 2146 0038 214C
03FDD8  0050 214E 005C 700A 7401 7600 4E41 4E75
03FDE8  43FA 000A 3478 00C8 4E92 4E75 0000 D007
03FDF8  0200 0100 0000 0000

03FE00  .... .... (Datenbereich mit Stapel usw.)
```

Monitorprogramms, die folgenden 54 (!) zu denen der erzeugten Tochterprogramme, der 56. nach MCLONE. Bitte beachten Sie, daß diese Vektoren mit MSB 00 (und nicht etwa FF) beginnen: Die Jobs wurden ja nicht aus dem Speicher entfernt, sondern lediglich auf Nullpriorität gesetzt.

Wegen der offenkundigen Gleichförmigkeit wurden von den 54 Töchtern nur zwei abgedruckt. Wie schon von den vorhergehenden Kapiteln her bekannt, gehören zu einem selbständig (unter der Kontrolle von QDOS) ablaufenden Programm drei Dinge: Der Job-Kontrollblock, das eigentliche Programm (Objektcode) und der Datenbereich, der hier nur den Stapel aufzunehmen hat.

37.2 Die Tochterprogramme

Das Listing P.37.1 (I) ist Rekonstruktion und Faksimile zugleich: Um den Vergleich mit dem Speicherauszug zu erleichtern, wurde es auf absolute Adressen umgestellt; um Nicht-QL-Usern entgegenzukommen, wurde der Motorola-Norm entsprechend das «(PC)» eingefügt. Schließlich beginnt das Listing mit einer *Rekonstruktion* des F-1-Programms, das in Wahrheit in *dieser* Form zu keinem Zeitpunkt existiert, sondern nur nach Durchlauf von CREA-JOB als umkopierter Objektcode. Der einzugebende Quelltext beginnt nach dem entsprechenden Hinweis.

F-1 (und alle anderen auch) besteht also nur aus dem Programmkopf, über den hinweg zu dem allen Programmen gemeinsamen Rumpf SCLONE gesprungen wird, der mit einer Desaktivierung endet – sozusagen also ein Sprung ohne Wiederkehr. Die in unserem Zusammenhang wichtigste Aufgabe von SCLONE ist die Erzeugung solcher Programmköpfe und ihre Anmeldung als selbständige Programme beim Betriebssystem.

In Abschnitt 34.2 wurde mitgeteilt, daß gemäß QDOS-Konvention die einen Job als selbständiges Programm ausweisende Kennzahl $4AFB im sechsten Byte (Zählung ab 0) des Programms beginnen muß. Hier wird nun deutlich, daß für die Formulierung des «Über-den-Kopf-hinweg-Sprungs» bis zu sechs Bytes freizuhalten sind.

◂ *Bild 37.1* Another self-cloning program – Speicherauszug nach Ablauf

```
                ****** Tochterprogramm F-1 *******
03FCB8  4EF9 0003 FD5A  FCLON1    JMP       SCLONE
03FCBE  4AFB                      DC.W      $4AFB
03FCC0  0006                      DC.W      6
03FCC2  5361 6263 7573            DC.B      'Sabcus'

                ************************************
                *     (ab hier ist einzugeben)     *
                ************************************

                          DIES      EQU       -1
                          MT_CJOB   EQU       1
                          MT_ACTIV  EQU       $0A
                          MT_PRIOR  EQU       $0B
                          MT_SUSJB  EQU       8
                          UT_SCR    EQU       $C8
                          IO_SBYTE  EQU       5
                          JB_D4     EQU       $30
                          JB_D5     EQU       $34
                          JB_D6     EQU       $38
                          JB_A4     EQU       $50
                          JB_A7     EQU       $5C
03FD38  600E                      START     BRA.S     MCLONE
03FD3A  0000 0000                 DS.B      4
03FD3E  4AFB                      DC.W      $4AFB
03FD40  0006                      DC.W      6
03FD42  4142 4375 7377            DC.B      'ABCusw'

03FD48  6100 009E         MCLONE  BSR       SCREEN
03FD4C  2848                      MOVEA.L   A0,A4
03FD4E  7A40                      MOVEQ     #'@',D5
03FD50  6100 0038                 BSR       CREAJOB
03FD54  2C01                      MOVE.L    D1,D6
03FD56  6000 002A                 BRA       FERTIG
```

P.37.1 Another self-cloning program (I)

37.3 Das Masterprogramm

Die EQU-Anweisungen und der Programmkopf sind reine Routinearbeit. Zum Eröffnen eines für alle Jobs gemeinsamen Fensters ist nur zu erwähnen, daß der von UT_SCR zurückgegebene channel-ID in A4 zwischengespeichert wird. D5 wird dann mit dem ASCII des Masterspace-Zeichens geladen.

Nach der Erzeugung eines neuen Programmkopfes, worauf wir schnell genauer eingehen wollen, wird die Job-Kennzahl des neuen Jobs nach D6 zwischengespeichert, und MCLONEs Priorität wird auf Null gesetzt.

37.4 Die Jobproduzierroutine CREAJOB

CREAJOB beginnt mit der Anmeldung eines neuen Jobs beim Betriebssystem. MT_CJOB *benötigt* (neben der Funktionsauswahl in D0) *folgende Parameter:*

In D1: Job-Kennzahl des anmeldenden Jobs. Eine negative Zahl besagt hier, daß der momentan aktive Job ein ihm nachgeordnetes Programm anmeldet.
In D2: Länge des Programmtextes; bitte zählen Sie im Hexdump bzw. im Pseudolisting nach.
In D3: Die Länge des Datenbereichs. Da in unserem Beispiel nur der Stapel unterzubringen ist, ist dataspace mit 8 überdimensioniert.
In A1: Wenn keine besondere Startadresse vorgeschrieben wird, ist A1 auf Null zu setzen.

MT_CJOB *gibt folgende Daten zurück:*

In D0: Eine evtl. Fehlermeldung ≠ 0. Wird eine solche festgestellt, geht es unverrichteterdinge zurück zum auf den Aufruf folgenden Befehl.
In D1: Die Job-Kennzahl (Job-ID) des neu angemeldeten Programms.
In A0: «base of area allocated», die Adresse, von der an das Programm (Objektcode) bereitzustellen ist.

A.37.1:

a) Welche Stapeleintragungen weisen F−1 und F−2 auf? Erklärung?
b) Wurden – unter der Voraussetzung zeitlicher Homogenität – F−1 bzw. F−2 während oder außerhalb der Abarbeitung von CREAJOB dokumentiert?

```
03FD5A  7005                SCLONE   MOVEQ    #IO_SBYTE,D0
03FD5C  1205                         MOVE.B   D5,D1
03FD5E  76FF                         MOVEQ    #-1,D3
03FD60  204C                         MOVEA.L  A4,A0
03FD62  4E43                         TRAP     #3
03FD64  4A80                         TST.L    D0
03FD66  661A                         BNE.S    FERTIG

03FD68  7008                SCL2     MOVEQ    #MT_SUSJB,D0
03FD6A  72FF                         MOVEQ    #DIES,D1
03FD6C  760A                         MOVEQ    #10,D3
03FD6E  93C9                         SUBA.L   A1,A1
03FD70  4E41                         TRAP     #1

03FD72  6100 0016                    BSR      CREAJOB
03FD76  7EFF                         MOVEQ    #-1,D7
03FD78  4E71                WARTEN   NOP
03FD7A  51CF FFFC                    DBRA     D7,WARTEN
03FD7E  51CC FFDA                    DBRA     D4,SCLONE

03FD82  700B                FERTIG   MOVEQ    #MT_PRIOR,D0
03FD84  72FF                         MOVEQ    #DIES,D1
03FD86  7400                         MOVEQ    #0,D2
03FD88  4E41                         TRAP     #1
                *****                          *****
                     **  Job inaktiviert   **
                      **********************
03FD8A  7001                CREAJOB  MOVEQ    #MT_CJOB,D0
03FD8C  72FF                         MOVEQ    #DIES,D1
03FD8E  7410                         MOVEQ    #16,D2
03FD90  7608                         MOVEQ    #8,D3
03FD92  93C9                         SUBA.L   A1,A1
03FD94  4E41                         TRAP     #1
03FD96  4A80                         TST.L    D0
03FD98  6702                         BEQ.S    NEWJOB
03FD9A  4E75                         RTS

03FD9C  2C48                NEWJOB   MOVEA.L  A0,A6
03FD9E  2C01                         MOVE.L   D1,D6
03FDA0  3CFC 4EF9                    MOVE.W   #$4EF9,(A6)+
03FDA4  43FA FFB4                    LEA      SCLONE(PC),A1
03FDA8  2CC9                         MOVE.L   A1,(A6)+
03FDAA  3CFC 4AFB                    MOVE.W   #$4AFB,(A6)+
03FDAE  2CFC 0006 5361               MOVE.L   #$00065361,(A6)+
03FDB4  2CBC 6263 7573               MOVE.L   #$62637573,(A6)
03FDBA  4DE8 0018           NJ1      LEA      24(A0),A6
```

P.37.1 Another self-cloning program (II)

Zur Erzeugung eines neuen Programms im Speicher gehören zwei Dinge: a) die Erzeugung eines Programmkopfes im Speicher; diesen Part übernimmt NEWJOB, b) die Bereitstellung der Anfangsdaten dort, von wo sie bei Aktivierung des Jobs durch den Scheduler in die Register geladen werden, also im Job-Kontrollblock. Das geschieht ab NJ2.

NEWJOB kopiert zunächst den Inhalt von A0 (nämlich den Beginn des neuen Programms) nach A6 und den Job-ID aus D1 nach D6. Dann werden mit Postinkrement folgende Bestandteile des neuen Programms an ihrem Ort abgelegt:

der Opcode von JMP, also $4EF9,
das Sprungziel SCLONE,
die Konventionszahl $4AFB und
der Name des Tochterprogramms, bestehend aus der Längenangabe 0006 und dem String ‹Sabcus›.

Abschließend wird bestimmt, wohin der Stapelzeiger des neuen Programms bei Erstaktivierung zeigen soll: 24 = 16 + 8, also nach Programmanfang + Programmlänge + Länge des Datenbereichs; A7 soll also bei Erstaktivierung unmittelbar «unter» den Stapel zeigen, der am Ende des Datenbereichs eingerichtet werden soll. Diese Adresse wird in A6 berechnet und für das Weitere festgehalten.

A.37.2: Sagen Sie voraus, ob das Programm mit LEA 22(A0),A6 oder mit LEA 26(A0),A6 auch ordnungsgemäß laufen würde.

Der Abschnitt NJ2 beginnt mit der Zuweisung der vorläufigen Priorität 0. MT_PRIOR benötigt *folgende Vorgaben:*

in D1 den Job-ID des hier besprochenen Programms; er steht seit MT_CJOB unverändert in D1,
in D2 die zugewiesene Priorität; hier (zunächst) 0. Das muß so sein, denn vor der Aktivierung müssen ja noch die Anfangsdaten bereitgestellt werden. Wozu dann überhaupt hier schon MT_PRIOR? Dieser Manager-TRAP liefert *folgende wesentlichen Rückgaben:*

in D0 Fehlermeldung; wir folgen dem Vorbild von A. Dickens und verzichten im Vertrauen auf die vorherige Fehlerabfrage durch MT_CJOB auf eine neue Prüfung;
in A0 die Basisadresse des Job-Kontrollblocks; sie dient nun mit Adressierungsart «Adreßregister indirekt mit Adreßdistanz» als Basis für die Zuwei-

```
03FDBE  700B            NJ2     MOVEQ   #MT_PRIOR,D0
03FDC0  7400                    MOVEQ   #0,D2
03FDC2  4E41                    TRAP    #1
03FDC4  217C 0000
        0008 0030               MOVE.L  #8,JB_D4(A0)
03FDCC  5285                    ADDQ.L  #1,D5
03FDCE  2145 0034               MOVE.L  D5,JB_D5(A0)
03FDD2  2146 0038               MOVE.L  D6,JB_D6(A0)
03FDD6  214C 0050               MOVE.L  A4,JB_A4(A0)
03FDDA  214E 005C               MOVE.L  A6,JB_A7(A0)

03FDDE  700A            NJ3     MOVEQ   #MT_ACTIV,D0
03FDE0  7401                    MOVEQ   #1,D2
03FDE2  7600                    MOVEQ   #0,D3
03FDE4  4E41                    TRAP    #1
03FDE6  4E75                    RTS
        *  ==================================
03FDE8  43FA 000A       SCREEN  LEA     SCRDEF(PC),A1
03FDEC  3478 00C8               MOVEA.W UT_SCR,A2
03FDF0  4E92                    JSR     (A2)
03FDF2  4E75                    RTS
03FDF4  0000 D007       SCRDEF  DC.L    $0000D007
03FDF8  0200 0100               DC.L    $02000100
03FDFC  0000 0000               DC.L    $00000000

                                END
```

P.37.1 Another self-cloning program (III)

sung der Anfangswerte. Auch in der Schreibweise dieser Adreßdistanzen folgen wir DICKENS. Bitte vergleichen Sie die am Programmanfang den Distanzen JB_xx zugewiesenen Werte mit der Übersicht in Abschnitt 35.2. Der Reihe nach werden nun bereitgestellt:

für das künftige D4 der Anfangsstand des Schleifenzählers, 8: Bis zu neunmal soll also jedes F-Programm seinen Kennbuchstaben ausgeben;
für das künftige D5 der um 1 erhöhte Inhalt des momentan benutzten D5. Die F-Programme *gerader* Linie sollen also 'A', 'B', 'C', ... ausgeben;
für das künftige D6 der Job-ID des neuen Programms;

für das künftige A4 die in A4 zwischengespeicherte, allen Jobs gemeinsame Kanalkennzahl;
für den neuen Stackpointer der mit LEA gewonnene und in A6 aufbewahrte Anfangswert.

(NB: Auch im Stapel lassen sich Daten an das neue Programm übergeben. SCLONE müßte dann mit dem Auslesen des Stapels beginnen.)

Erst nachdem das alles vorbereitet ist, gibt MT_ACTIV den neuen Job zum Ablauf frei, indem er die Priorität auf 1 setzt.

37.5 Der Gesamtablauf

Das mit exec gestartete MCLONE macht also das Fenster auf, erzeugt ein Tochterprogramm und begibt sich zur Ruhe. Der Scheduler aktiviert das F-1-Programm. Es springt über seinen Kopf hinweg nach SCLONE, gibt seinen Kennbuchstaben 'A' aus und suspendiert sich für eine fünftel Sekunde (10 Wechselstromzyklen). Wieder aktiviert, erzeugt es ein F-2-Programm, verharrt einen kurzen Augenblick in einer Warteschleife und kehrt noch (bis zu) achtmal nach SCLONE zurück.

A.37.3: Wieso?

Das vorübergehende Suspendieren ermöglicht die Zuweisung von Rechenzeit an die nach und nach entstehenden Tochterprogramme, die ihrerseits auf dem gleichen Wege neue Clones erzeugen, die ihren Kennbuchstaben ausgeben, wenn möglich einen neuen Job erzeugen, das mehrmals, bis sie sich schließlich selbst desaktivieren.

Die Ausgabe auf dem Bildschirm verblüfft zunächst. Bei einem Probelauf begann sie «ABCDBECFCDGD...» und endete «...OFFOJJFIIKF», mit dem 486. Zeichen. Die Häufigkeit der ausgegebenen Zeichen war

A: 1	D: 19	G: 46	J: 61	M: 26
B: 2	E: 45	H: 70	K: 35	N: 9
C: 4	F: 71	I: 62	L: 18	O: 17, zus. 486.

Feststellungen:

☐ Die Abläufe sind nicht immer reproduzierbar; ein und dasselbe Programm kann leicht unterschiedliche Buchstabenfolgen erbringen.

- ☐ Weder das A noch B oder C wurden neunmal ausgegeben, F dagegen sogar 71mal.
- ☐ Der hier nicht wiedergegebene Speicherauszug zeigt kein A, B oder C ausgebendes Programm!

A.37.4: Gibt es hierfür Erklärungen?

Die wesentlichen Aspekte des self-cloning sind als Möglichkeiten des Prozessors M68000 wohl hinreichend deutlich geworden. Da die Feinheiten des Betriebssystems QDOS nicht Gegenstand dieses Buches sind, gehen wir nicht weiter in die Tiefe. Interessierte QL-Besitzer finden hier aber auch ein reiches Experimentierfeld: Wie kann man jedem F-Job (zum Zwecke der Identifizierung) im Programmkopf einen individuellen Namen zuweisen? Wie ließen sich diese unverwechselbar codieren? Wie lassen sich Daten via Stapel übergeben? Welchen Einfluß haben andere Verzögerungen (Warteschleife, Suspendierung)?

38
The self-cloning program von Adrian DICKENS

Die Blattlose Euphorbie, ein Wolfsmilchgewächs, gleicht einer mit grünem Kunststoff beschichteten Drahtplastik. Alle fingerbreit teilt sich der Sproß, aus einem werden zwei, aus zweien werden vier, bis einmal einer abbricht.

Adrian DICKENS hat diese Struktur unübertrefflich elegant auf den Bildschirm gebracht, wobei das Besondere ist, daß jeder Sproß von einem selbständigen Programm erzeugt wird, das wiederum selber neue Programme erzeugt.

Das Spektakel beginnt in der Mitte des Bildschirms. Zwei Random-Zahlen bestimmen die Farbe und die Richtung des neuen Sprosses (Bild 38.1).

Interessierte Leser finden dieses Programm in dem Buch von A. DICKENS (s. Literaturverzeichnis) oder als kommentierten Abdruck in CHIP 9-86 S. 192.

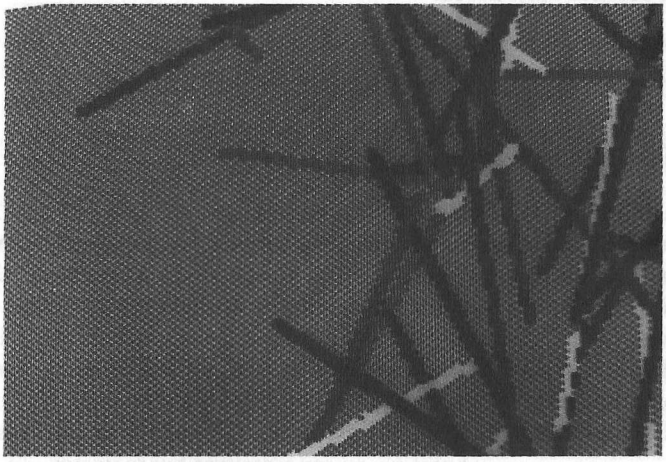


Bild 38.1 The self-cloning program

39
Die unvollendete Raummühle (für den NDR-Computer)

Raummühle wird in einem aus 4 × 4 × 4 = 64 Teilwürfeln (kurz «Felder») zusammengesetzten Würfel (kurz «Spielfeld») gespielt. Mensch und Computer setzen abwechselnd ihre M- oder C-Steine. Wer als erster vier Steine in gerader Linie («Mühle») plazierte, hat gewonnen. Eine Mühle kann parallel zu einer Kante oder Flächendiagonale verlaufen, kann aber auch einer Raumdiagonale folgen.

Hat einer der beiden Partner gewonnen, wird das angezeigt, und der Mensch kann wählen, ob er einen neuen Durchgang wünscht oder ins Betriebssystem zurückkehren möchte. Ein Unentschieden ist noch nicht eingebaut; insofern ist das vorgestellte Programm noch unvollendet. An den interessierten Leser ergeht die Anregung, das Fehlende nachzuholen, was nicht sonderlich schwer sein wird. Er wird aber vorher schon viel Spaß mit diesem schönen Spiel haben.

39.1 Die Darstellung auf dem Bildschirm

Das Spielfeld wird auf dem Bildschirm in vier Schichten («Ebenen») dargestellt (Bild 39.1), Ebene 1 ist die vorderste, die dem Menschen zugewandte Würfelschicht mit den Feldern $00 ... $0F. Ebene 2 mit den Feldern $10 ... $1F ist daneben abgebildet, liegt aber in Wahrheit dahinter, gefolgt von den Ebenen 3 und 4. Das Bemühen, sich den in die Ebene projizierten Ablauf räumlich vorzustellen, trainiert das Raumvorstellungsvermögen recht wirksam.

Will der Mensch nun das dritte Feld in der zweiten Zeile der Ebene 4 besetzen, muß er die Eingabeaufforderung E Z S (Ebene, Zeile, Spalte) mit 4 2 3 beantworten. Wichtig ist, daß genau nach Vorgabe formatiert eingege-

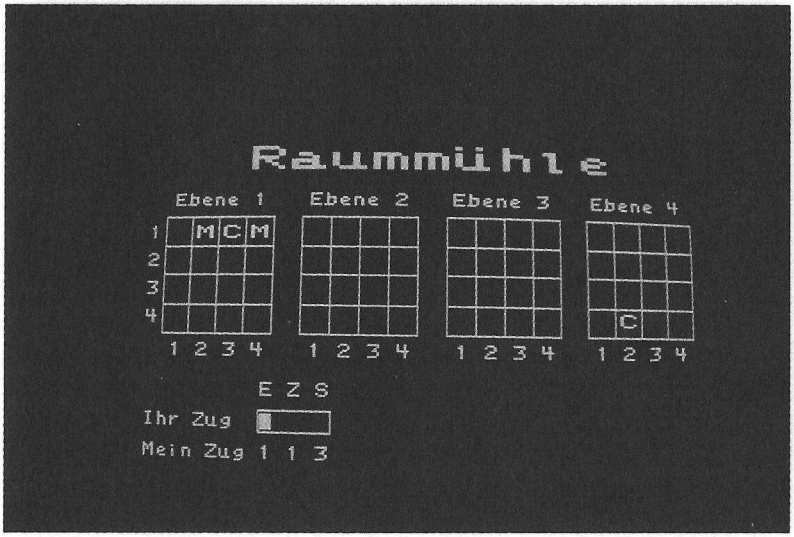


Bild 39.1 Die Darstellung auf dem Bildschirm

ben wird: 4 sp 2 sp 3, wobei sp die Leertaste symbolisiert. Fehlerhafte Eingaben werden ignoriert, können aber auch zu unerwarteten Reaktionen führen.

Die Fleißarbeit der grafischen Gestaltung der Spieloberfläche braucht nicht näher erläutert zu werden, doch werden für die Durchsicht des Quelltextes folgende Hinweise hilfreich sein:

WAGLIx: Diese Routinen zeichnen die WAaGerechten LInien des Spielfeldes. Ihr jeweiliger Ursprung wird in D3 errechnet. HOLDAT splittet den errechneten Vektor in seine Komponenten auf und weist sie D1 und D2 zu.
SENLIx: Ganz entsprechend werden die SENkrechten LInien hervorgebracht.
ZAHLEW: Hier werden die waagerecht unter dem Spielplan stehenden Zahlen ausgegeben;
ZAHLES: entsprechend senkrecht am linken Rand.
ORAND: Ausgabe der oberen Randleiste (Ebene 1 ...).

39.2 Interne Codierung

An diesem Spiel wird die Komplementarität von Mensch und Computer deutlich:

Phantasie: Der Mensch kann sich immer neue Strategien ausdenken; der Computer kann nur die nachvollziehen, die im Programm vorgegeben wurde.

Sicherheit: Die Konzentrationsfähigkeit des Menschen ist begrenzt; wer kann schon 76 mögliche und dazu noch räumliche Mühlen zuverlässig auf Gefahr und Chance hin absuchen? Der Computer kann's, wenn auch nicht mit absoluter Sicherheit (man denke an Stromausfall und Störwellen).

Schnelligkeit: Wer einmal Raummühle in BASIC spielte, wird sich an die erforderliche Geduld erinnern. Ein Maschinenprogramm ist 200- bis 300mal schneller, so daß der Computer scheinbar «keine Denkpause» benötigt. Trotz der stupiden Prüfung aller Möglichkeiten – auch der unwesentlich gewordenen – reagiert er sehr viel schneller als der Mensch.

Diese Komplementarität bestimmt die Strategie, und diese die interne Codierung:

Parallel zur Spieloberfläche wird im Speicher ein Spielfeld von 64 Bytes eingerichtet und zur Erleichterung einer Kontrolle des Spielstandes (z.B. während der Fortschreibung des Programms) mit 'SPIELF' markiert. Jedes Byte trägt zwei Codes – das LSN die Feldbesetzung, das MSN den Spielwert.

Die Codierung des LSN bedeutet

1: Das Feld ist unbesetzt;
2: das Feld ist vom Menschen besetzt;
3: das Feld ist vom Computer besetzt.

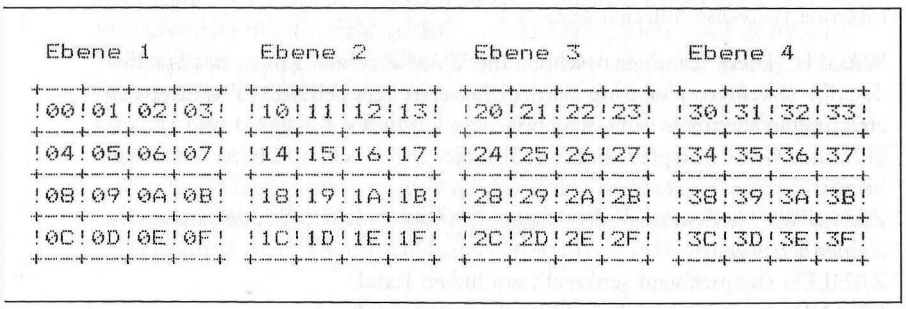


Bild 39.2 Die Numerierung der Felder

Zur Herleitung des Spielwertes müssen wir weiter ausholen. Bild 39.2 zeigt die Numerierung der Felder (Elementarwürfel). Durch jedes Feld laufen (mindestens) vier mögliche Mühlen, drei in Richtung der Würfelkanten und eine parallel zu einer Flächendiagonale. Die acht äußersten und die acht innersten Würfel werden von drei weiteren (möglichen) Mühlen durchlaufen: Hier treffen sich drei Parallelen zu Flächendiagonalen und eine Raumdiagonale.

Leser, die Strategie und Ausführung verstehen wollen, sollten die Tabelle MLDUFN (MühLen DUrch Feld-Nummer) am Ende des Programms beachten; sie ordnet den 64 Feldern des Spielwürfels je vier oder sieben Mühlen zu, die von $00 ... $4B numeriert sind. (Wie gut, daß es für derartige Fleißarbeiten Computer gibt!) Die negativen Eintragungen formatieren lediglich zu einheitlichen Achtergruppen, werden aber bei der Auswertung ignoriert.

A.39.1: Beschreiben Sie den Verlauf der Mühlen $20, $35 und $49.

Für jede dieser (48 * 4 + 16 * 7) : 4 = 76 möglichen Mühlen wird ein «Mühlenprodukt» ermittelt. Ist z.B. ein Feld einer (möglichen) Mühle frei, eins vom Computer besetzt und zwei vom Menschen, so sind die entsprechenden LSNs 1, 3, 2, 2, und ihr Produkt, das Mühlenprodukt, beträgt 12. Umgekehrt beschreibt das Mühlenprodukt 12 die Besetzung der Mühle, allerdings ohne die irrelevante innere Anordnung (1223, 2123, 3122 ... sind in diesem Sinne gleichwertig).

Die Mühlenprodukte sind noch kein unmittelbares Maß für den Spielwert. Dem hohen Mühlenprodukt 18 kommt beispielsweise nur ein geringer Spielwert zu, denn auf dieser Mühle ist nichts mehr zu gewinnen; als einzige Möglichkeit bleibt die (wirkungslose) Besetzung des freien Feldes.

WERTTA ordnet den Mühlenprodukten Spielwerte nach folgender Wertung zu:

MP ← Mühlenprodukt				Spielwert → SW
27		3C	Zug und Sieg für den Computer	9
8	3M		höchste Gefahr, Mensch vor Sieg	8
			(Gefahr einer Zwickmühle	7)
9		2C	Computer kann Chance ausbauen	6
4	2M		hier bahnt sich eine Gefahr an	5
3		1C	Chance für den Computer in Sicht	4
2	1M		hier könnte eine Gefahr entstehen	3

1			unbesetzt; man könnte aufbauen	2
6	1M	1C	hier keine Entscheidung möglich	1
12	2M	1C	hier keine Entscheidung möglich	1
18	1M	2C	hier keine Entscheidung möglich	1
24	3M	1C	Tetrade ohne Entscheidung besetzt	0
36	2M	2C	Tetrade ohne Entscheidung besetzt	0
54	1M	3C	Tetrade ohne Entscheidung besetzt	0
16	4M		Siegmeldung Mensch	
81		4C	Siegmeldung Computer	

Da also einem freien Feld einerseits der Spielwert 2 zuerkannt und andererseits der Frei-Status mit 1 im LSN zu codieren ist, muß SPIELF mit $21 initialisiert werden, was bei INIFLD geschieht. Außerdem wird noch eine Tabelle der Mühlenprodukte (PROTAB) benötigt, die mit Anfangswert 01 zu belegen ist, und eine Tabelle der Spielwerte (TABSPW) mit Anfangswert 02 für «freies Feld». Auch diese Feldnamen stehen als String im Speicher (Bild 39.3).

Der Hexdump von Bild 39.3 wurde nach folgendem Spielverlauf gezogen:

M: 1 1 2, C: 4 4 2, M: 1 1 4, C: 1 1 3, Abbruch.

Den Spielstand zeigt Bild 39.1. Man erkennt in SPIELF in vielen Feldern die Anfangsbelegung $21. Ferner ist zu erkennen, daß die Felder $01 und $03

```
0207F2: 53 50 49 45 4C 46 = 'SPIELF'
0207F8: 51525352 21313131 21312131 31412131 (Ebene 1)
020808: 21313131 21213131 21212121 21412121 (Ebene 2)
020818: 21312131 21212121 21312131 21412121 (Ebene 3)
020828: 31412131 21412121 21412121 41434141 (Ebene 4)

020838: 50 52 4F 54 41 41 = 'PROTAB'
02083E: 01020302 01010101 01010101 01030101 (vo - hi)
02084E: 0C010101 01010101 01010101 01010103 (li - re)
02085E: 01020302 01010101 01010101 01030101 (ob - un)
02086E: 01010101 02010101 ........          (........)

02088A: 54 41 42 53 50 57 = 'TABSPW'
020890: 02030203 02020202 02020202 02040202 (Ebene 1)
0208A0: ........
```

Bild 39.3 Raummühle, ein Blick in den Speicher

vom Menschen und $02 und $3D vom Computer besetzt sind. Offenbar war *vor* dem 113-Zug des Computers die horizontale Tetrade (in der vorderen oberen Kante) die sensibelste; sie hat die Nummer $10, und ihr Mühlenprodukt *war* 04 (und ist nach dem Zug $0C); dem entspricht der in vier MSNs abgelegte Spielwert 5.

Die Mühlenprodukte werden bei jedem neuen Zug fortgeschrieben. Wird ein freies Feld vom Menschen besetzt, werden die Produkte aller durch dieses Feld verlaufenden Mühlen verdoppelt, beim Computer verdreifacht, was MULPRO bewerkstelligt. Um diese Routine sowohl für M- als auch für C-Züge benutzen zu können, wird vorher die Nummer des neu besetzten Feldes in FELDNR und der entsprechende Faktor 2 oder 3 in FAKTOR bereitgestellt.

39.3 Zwickmühlen

Der Begriff «Zwickmühle» wurde dem klassischen Mühlespiel entlehnt. Er bezeichnet hier die Situation, daß durch das Setzen eines einzigen Steins zwei Mühlen «aufgemacht» werden. Da der Gegner, falls er seinerseits keine offene Mühle schließen kann, nur eine blockieren kann, führt die andere zum Sieg. Unser Programm läßt den Computer zwar nicht zielstrebig Zwickmühlen vorbereiten (dazu gehörte schon ziemlich viel «Phantasie»), aber es schützt ihn durch sorgfältige Suche nach den Anfängen dieser gefährlichen Konstellation.

Eine Zwickmühle entsteht, wenn der Mensch ein Feld besetzt, durch das zwei Tetraden mit Spielwert 5 verlaufen. Einem solchen Feld wird der Spielwert 7 zugeordnet. Diese Bewertung dürfte mit Blick auf die abgedruckte Tabelle plausibel sein. Das Label ZWICKM verrät die Aufgabe der hier beginnenden Routine.

39.4 Die Strategie

Die vom Computer verfolgte Strategie ist bereits im wesentlichen umrissen. Der Programmabschnitt ZUGCOM sucht im vierten Abschnitt SUCHZG das als nächstes vom Computer zu besetzende Feld, das mit dem höchsten Spielwert. Man könnte eine solche Strategie mit einer gewissen Berechtigung als reaktiv/defensiv bezeichnen, wird aber bald merken, daß der Computer dank seiner dominanten «Aufmerksamkeit» ganz schön offensiv werden kann.

SUCHZG beginnt zunächst einmal mit der Suche nach dem freien Feld mit dem höchsten Spielwert (SUCHMX).

A.39.2: A2, A3 und A4 sind (nahezu globale) Zeiger nach TABSPW, SPIELF bzw. MLDUFN. Welches Register enthält beim Verlassen von SUCHMX in Bit 7 ... 4 den maximalen Spielwert?

Nun wird der Mensch erheblich verunsichert: Der Computer reagiert auf gleiche Situationen nicht mit demselben Zug. Er sucht nun die Nummer des Feldes mit dem höchsten Spielwert; deren kann es aber mehrere geben. Nach Maßgabe einer Random-Zahl sucht unser Gegenspieler SPIELF mal vorwärts (SUCHVW) und mal rückwärts (SUCHRW) nach diesem Feld ab.

ZEIGZC zeigt nun den Zug des Computers im E Z S-Schema an.

A.39.3: Beschreiben Sie die wesentlichen Schritte der Aufbereitung der Feldnummer zur Anzeige.

SETZZC bringt schließlich das Symbol C in der richtigen Position in die Spieloberfläche. Da das C praktisch schon dann erscheint, wenn der Mensch noch auf seine Eingabe konzentriert ist, wäre es sehr mühsam, jeden Computerzug anhand der angezeigten Zahlen lokalisieren zu müssen. Die sich einige Male wiederholende Ausgabe Schmal-C, Mittel-C, Breit-C (FLACKE) hilft dem optisch ab.

39.5 Die Siegmeldung

Das Hauptprogramm ruft nach jedem ZUGMEN und jedem ZUGCOM die Subroutine OBSIEG auf, deren Aufgabe nicht schwer zu erraten sein wird. Das Register D6 trat bisher nicht in Erscheinung. Die eigentliche Multiplikation bei der Fortschreibung der Mühlenprodukte erledigt MULTIM. Wird dabei MP = 16 (Sieg Mensch) festgestellt, wird dieses charakteristische Produkt nach D6 übergeben, im Falle 81 (Sieg Computer) der Code 17. OBSIEG fragt D6 danach ab und veranlaßt ggf. die Ausgabe einer entsprechenden Meldung.

Da es oft schwierig ist, aus der letzten Zugmeldung des Computers seine Siegposition zu erkennen, wird diese durch zusätzliche Umrahmung der vier Cs innerhalb ihrer Felder hervorgehoben. Hieran sind die Routinen ZEISIG

(zeig die Siegposition), ZEIKAS (zeichne einen Kasten), KASTN, QUASEI (zwei Quadrat-, in Wahrheit Rechteckseiten) und QUAKL (kleinere Quadratseiten) beteiligt.

39.6 Weiterer Ausbau

Der Phantasie sind (fast) keine Grenzen gesetzt. Eine naheliegende Erweiterung wäre, wie schon eingangs angeregt, die Feststellung, daß keiner der beiden Spieler mehr siegen kann. Man könnte das auch so ausspinnen: «Nehmen Sie ein Unentschieden an?» Lehnt der Mensch ab, *muß* er siegen; anderenfalls hätte er aufgrund seiner Ablehnung verloren. Man könnte auch anzeigen «Rien ne va plus», wenn alle Felder ergebnislos besetzt sind.

```
020000  6100 0014          START:  BSR      INIT
020004  6100 0288          ZUGMEN: BSR      EINGAB
020008  6100 0420                  BSR      OBSIEG
02000C  6100 05AA                  BSR      ZUGCOM
020010  6100 0418                  BSR      OBSIEG
020014  60EE                       BRA.S    ZUGMEN
020016
020016  2E0F               INIT:   MOVE.L   A7,D7
020018  7C00                       MOVEQ    #0,D6
02001A  6100 000C                  BSR      SCREEN
02001E  6100 001A                  BSR      BILD
020022  6100 0234                  BSR      INIFLD
020026  4E75                       RTS
020028
020028  4EB9 00003FEC      SCREEN: JSR      @CLRSCREEN
02002E  7000                       MOVEQ    #0,D0
020030  7201                       MOVEQ    #1,D1
020032  4EB9 00001F06              JSR      @SETFLIP
020038  4E75                       RTS
02003A
02003A  4EB9 00001342      BILD:   JSR      @FIRSTTIME
020040  6100 001C                  BSR      WAGLIS
020044  6100 0078                  BSR      SENLIS
020048  6100 00C8                  BSR      ZAHLEW
02004C  6100 0116                  BSR      ZAHLES
020050  6100 0144                  BSR      ORAND
```

```
020054  6100 0184              BSR     TITEL
020058  6100 01A2              BSR     EZS
02005C  4E75                   RTS
02005E  263C 01200012  WAGLIS: MOVE.L  #$01200012,D3
020064  7000                   MOVEQ   #0,D0
020066  4EB9 0000121E          JSR     @AUFK
02006C  6100 0010              BSR     WAGLI3
020070  6100 0002              BSR     WAGLI1
020074  6100 0002      WAGLI1: BSR     WAGLI2
020078  0683 001F0000  WAGLI2: ADDI.L  #$001F0000,D3
02007E  4EB9 0000108A  WAGLI3: JSR     @HEBE
020084  6100 002E              BSR     HOLDAT
020088  4EB9 0000122E          JSR     @AUFXY
02008E  6100 0002              BSR     WAGLI6
020092  6100 0002      WAGLI6: BSR     WAGLI7
020096  4EB9 00001090  WAGLI7: JSR     @SENKE
02009C  7064                   MOVEQ   #100,D0
02009E  4EB9 00001288          JSR     @SCHREITE
0200A4  4EB9 0000108A          JSR     @HEBE
0200AA  7019                   MOVEQ   #25,D0
0200AC  4EB9 00001288          JSR     @SCHREITE
0200B2  4E75                   RTS
0200B4  3203            HOLDAT: MOVE.W  D3,D1
0200B6  4843                   SWAP    D3
0200B8  3403                   MOVE.W  D3,D2
0200BA  4843                   SWAP    D3
0200BC  4E75                   RTS
0200BE
0200BE  263C 01200012  SENLIS: MOVE.L  #$01200012,D3
0200C4  705A                   MOVEQ   #90,D0
0200C6  4EB9 0000121E          JSR     @AUFK
0200CC  6100 001E              BSR     SENLI5
0200D0  6100 0012              BSR     SENLI3
0200D4  6100 0012              BSR     SENLI4
0200D8  6100 0002              BSR     SENLI1
0200DC  6100 0002      SENLI1: BSR     SENLI2
0200E0  6100 0002      SENLI2: BSR     SENLI3
0200E4  6100 0002      SENLI3: BSR     SENLI4
0200E8  0643 0019      SENLI4: ADDI.W  #25,D3
0200EC  4EB9 0000108A  SENLI5: JSR     @HEBE
0200F2  6100 FFC0              BSR     HOLDAT
0200F6  4EB9 0000122E          JSR     @AUFXY
0200FC  4EB9 00001090          JSR     @SENKE
020102  707C                   MOVEQ   #$7C,D0
020104  4EB9 00001288          JSR     @SCHREITE
02010A  4EB9 0000108A          JSR     @HEBE
```

```
020110  4E75                        RTS
020112
020112  263C 0084001A   ZAHLEW:  MOVE.L  #$0084001A,D3
020118  41FA 0074                 LEA     ZAHLTA(PC),A0
02011C  6100 0028                 BSR     ZAHLEB
020120  263C 00840097             MOVE.L  #$00840097,D3
020126  41FA 0066                 LEA     ZAHLTA(PC),A0
02012A  6100 001A                 BSR     ZAHLEB
02012E  263C 00840114             MOVE.L  #$00840114,D3
020134  41FA 0058                 LEA     ZAHLTA(PC),A0
020138  6100 000C                 BSR     ZAHLEB
02013C  263C 00840191             MOVE.L  #$00840191,D3
020142  41FA 004A                 LEA     ZAHLTA(PC),A0
020146  6100 0002   ZAHLEB:  BSR     ZAHLW1
02014A  6100 0002   ZAHLW1:  BSR     ZAHLW2
02014E  7021        ZAHLW2:  MOVEQ   #$21,D0
020150  6100 FF62                 BSR     HOLDAT
020154  4EB9 00001386             JSR     @WRITE
02015A  0643 0019                 ADDI.W  #25,D3
02015E  41E8 0002                 LEA     2(A0),A0
020162  4E75                        RTS
020164  263C 00C20002   ZAHLES:  MOVE.L  #$00C20002,D3
02016A  41FA 0022                 LEA     ZAHLTA(PC),A0
02016E  6100 0002                 BSR     ZAHLS1
020172  6100 0002   ZAHLS1:  BSR     ZAHLS2
020176  7021        ZAHLS2:  MOVEQ   #$21,D0
020178  6100 FF3A                 BSR     HOLDAT
02017C  4EB9 00001386             JSR     @WRITE
020182  0483 000F0000             SUBI.L  #$000F0000,D3
020188  41E8 0002                 LEA     2(A0),A0
02018C  4E75                        RTS
02018E
02018E  3100        ZAHLTA:  DC.W    $3100
020190  3200                    DC.W    $3200
020192  3300                    DC.W    $3300
020194  3400                    DC.W    $3400
020196
020196  263C 00D4001A   ORAND:  MOVE.L  #$00D4001A,D3
02019C  41FA FFF0                 LEA     ZAHLTA(PC),A0
0201A0  43FA 0032                 LEA     TEXT(PC),A1
0201A4  6100 0002                 BSR     ORAN1
0201A8  6100 0002   ORAN1:  BSR     ORAN2
0201AC  7021        ORAN2:  MOVEQ   #$21,D0
0201AE  6100 FF04                 BSR     HOLDAT
0201B2  C149                      EXG     A0,A1
0201B4  4EB9 00001386             JSR     @WRITE
```

```
0201BA  0643 004A              ADDI.W  #$004A,D3
0201BE  6100 FEF4              BSR     HOLDAT
0201C2  C149                   EXG     A0,A1
0201C4  4EB9 00001386          JSR     @WRITE
0201CA  41E8 0002              LEA     2(A0),A0
0201CE  0643 0032              ADDI.W  #$32,D3
0201D2  4E75                   RTS
0201D4  4562656E6500    TEXT:  DC.B    'Ebene',0
0201DA
0201DA  303C 0062       TITEL: MOVE.W  #$62,D0
0201DE  323C 0060              MOVE.W  #$60,D1
0201E2  343C 00E8              MOVE.W  #$E8,D2
0201E6  41FA 000A              LEA     RAUMUE(PC),A0
0201EA  4EB9 00001386          JSR     @WRITE
0201F0  4E75                   RTS
0201F2  5261756D6DFD68  RAUMUE: DC.B    'Raumm}hle',0
0201F9  6C6500
0201FC  303C 0021       EZS:   MOVE.W  #$21,D0
020200  323C 006E              MOVE.W  #$6E,D1
020204  343C 0070              MOVE.W  #$70,D2
020208  41FA 0036              LEA     TXTEZS(PC),A0
02020C  4EB9 00001386          JSR     @WRITE
020212  303C 0021              MOVE.W  #$21,D0
020216  323C 0002              MOVE.W  #$02,D1
02021A  343C 0060              MOVE.W  #$60,D2
02021E  41FA 0026              LEA     TIHRZ(PC),A0
020222  4EB9 00001386          JSR     @WRITE
020228  303C 0021              MOVE.W  #$21,D0
02022C  323C 0002              MOVE.W  #$02,D1
020230  343C 0050              MOVE.W  #$50,D2
020234  41FA 0018              LEA     TMEINZ(PC),A0
020238  4EB9 00001386          JSR     @WRITE
02023E  4E75                   RTS
020240  45205A205300   TXTEZS: DC.B    'E Z S',0
020246  496872205A7567 TIHRZ:  DC.B    'Ihr Zug',0
02024D  00
02024E  4D65696E205A75 TMEINZ: DC.B    'Mein Zug ',0
020255  672000
020258
020258  47FA 059E       INIFLD: LEA     SPIELF(PC),A3
02025C  700F                   MOVEQ   #15,D0
02025E  26FC 21212121  LSPI21: MOVE.L  #$21212121,(A3)+
020264  51C8 FFF8              DBRA    D0,LSPI21
020268  47FA 05D4              LEA     PROTAB(PC),A3
02026C  7012                   MOVEQ   #18,D0
02026E  26FC 01010101  LPROT1: MOVE.L  #$01010101,(A3)+
```

```
020274  51C8 FFF8              DBRA    D0,LPROT1
020278  47FA 0616              LEA     TABSPW(PC),A3
02027C  7012                   MOVEQ   #18,D0
02027E  26FC 02020202  LSPWT:  MOVE.L  #$02020202,(A3)+
020284  51C8 FFF8              DBRA    D0,LSPWT
020288  47FA 056E              LEA     SPIELF(PC),A3
02028C  4E75                   RTS
02028E
02028E                         * == EINGABE MENSCH == *
02028E
02028E  41FA 0152      EINGAB: LEA     FAKTOR(PC),A0
020292  30BC 0002              MOVE.W  #2,(A0)
020296  6100 0014              BSR     EINLES
02029A  6100 004C              BSR     EINFLD
02029E  6100 0086              BSR     FLDFRE
0202A2  6100 0140              BSR     MULPRO
0202A6  6100 03C2              BSR     ZWICKM
0202AA  4E75                   RTS
0202AC  303C 0021      EINLES: MOVE.W  #$21,D0
0202B0  323C 006E              MOVE.W  #$6E,D1
0202B4  343C 0060              MOVE.W  #$60,D2
0202B8  7605                   MOVEQ   #5,D3
0202BA  41FA 0026              LEA     EBER(PC),A0
0202BE  4290                   CLR.L   (A0)
0202C0  4268 0004              CLR.W   4(A0)
0202C4  4EB9 00001486          JSR     @READ
0202CA  41FA 0016              LEA     EBER(PC),A0
0202CE  1168 0002 0001         MOVE.B  2(A0),1(A0)
0202D4  1168 0004 0002         MOVE.B  4(A0),2(A0)
0202DA  117C 0002 0004         MOVE.B  #2,4(A0)
0202E0  4E75                   RTS
0202E2                 EBER:   DS.B    5
0202E7                 FELDNR: DS.B    1
0202E8  203A FFF8      EINFLD: MOVE.L  EBER(PC),D0
0202EC  0480 01010100          SUBI.L  #$01010100,D0
0202F2  7200                   MOVEQ   #0,D1
0202F4  6100 0010              BSR     EINZUL
0202F8  6100 000C              BSR     EINZUL
0202FC  6100 0008              BSR     EINZUL
020300  1141 0005              MOVE.B  D1,5(A0)
020304  4E75                   RTS
020306
020306  E198          EINZUL:  ROL.L   #8,D0
020308  0C00 0030              CMPI.B  #$30,D0
02030C  6510                   BCS.S   UNZUL
02030E  0C00 0034              CMPI.B  #$34,D0
```

```
020312  6A0A                        BPL.S   UNZUL
020314  0200 0003                   ANDI.B  #3,D0
020318  E541                        ASL.W   #2,D1
02031A  D200                        ADD.B   D0,D1
02031C  4E75                        RTS
02031E  4FEF 0008           UNZUL:  LEA     8(A7),A7
020322  6000 FF6A                   BRA     EINGAB
020326
020326  1433 1000           FLDFRE: MOVE.B  0(A3,D1.W),D2
02032A  0202 0003                   ANDI.B  #3,D2
02032E  0C02 0001                   CMPI.B  #1,D2
020332  6608                        BNE.S   FLDBES
020334  0633 0001 1000              ADDI.B  #1,0(A3,D1.W)
02033A  6062                        BRA.S   ZEIGZM
02033C  41FA 0028           FLDBES: LEA     TXTBES(PC),A0
020340  6116                        BSR.S   FTXAUS
020342  41FA 003E                   LEA     TXTWEG(PC),A0
020346  701E                        MOVEQ   #30,D0
020348  4EB9 00000E5A              JSR     @DELAY
02034E  6108                        BSR.S   FTXAUS
020350  4FEF 0004                   LEA     4(A7),A7
020354  6000 FF38                   BRA     EINGAB
020358  7032                FTXAUS: MOVEQ   #$32,D0
02035A  720A                        MOVEQ   #$0A,D1
02035C  7428                        MOVEQ   #$28,D2
02035E  4EB9 00001386               JSR     @WRITE
020364  4E75                        RTS
020366  4461732046656C      TXTBES: DC.B    'Das Feld '
02036D  6420
02036F  69737420736368              DC.B    'ist schon'
020376  6F6E
020378  20626573657474A            DC.B    ' besetzt!',0
02037F  742100
020382  202020202020202     TXTWEG: DC.B    '               '
020389  2020
02038B  202020202020202             DC.B    '               '
020392  2020
020394  202020202020202             DC.B    '               ',0
02039B  202000
02039E
02039E  41FA FF42           ZEIGZM: LEA     EBER(PC),A0
0203A2  1428 0001                   MOVE.B  1(A0),D2
0203A6  0242 0007                   ANDI.W  #7,D2
0203AA  C4FC 000F                   MULU    #15,D2
0203AE  0442 00D1                   SUBI.W  #$D1,D2
0203B2  4442                        NEG.W   D2
```

```
0203B4  1010                        MOVE.B  (A0),D0
0203B6  0240 0007                   ANDI.W  #7,D0
0203BA  C0FC 0005                   MULU    #5,D0
0203BE  1228 0002                   MOVE.B  2(A0),D1
0203C2  0241 0007                   ANDI.W  #7,D1
0203C6  D240                        ADD.W   D0,D1
0203C8  C2FC 0019                   MULU    #$19,D1
0203CC  0441 007F                   SUBI.W  #$7F,D1
0203D0  7031                        MOVEQ   #$31,D0
0203D2  41FA 000A                   LEA     SYMMEN(PC),A0
0203D6  4EB9 00001386               JSR     @WRITE
0203DC  4E75                        RTS
0203DE  4D00                SYMMEN: DC.B    'M',0
0203E0  4300                SYMCOM: DC.B    'C',0
0203E2                      FAKTOR: DS.W    1
0203E4  123A FF01           MULPRO: MOVE.B  FELDNR(PC),D1
0203E8  4881                        EXT.W   D1
0203EA  E741                        ASL.W   #3,D1
0203EC  363A FFF4                   MOVE.W  FAKTOR(PC),D3
0203F0  45FA 044C                   LEA     PROTAB(PC),A2
0203F4  49FA 04EC                   LEA     MLDUFN(PC),A4
0203F8  41F4 1000                   LEA     0(A4,D1.W),A0
0203FC  1018                LMULPR: MOVE.B  (A0)+,D0
0203FE  6B06                        BMI.S   PRODEX
020400  6100 0006                   BSR     MULTIM
020404  60F6                        BRA.S   LMULPR
020406  4E75                PRODEX: RTS
020408
020408  4880                MULTIM: EXT.W   D0
02040A  1432 0000                   MOVE.B  0(A2,D0.W),D2
02040E  4882                        EXT.W   D2
020410  C4C3                        MULU    D3,D2
020412  0C42 0010                   CMPI.W  #16,D2
020416  6602                        BNE.S   NOTSM
020418  7C10                        MOVEQ   #16,D6
02041A  0C42 0051           NOTSM:  CMPI.W  #81,D2
02041E  6604                        BNE.S   NOTSC
020420  7C11                        MOVEQ   #17,D6
020422  1A00                        MOVE.B  D0,D5
020424  1582 0000           NOTSC:  MOVE.B  D2,0(A2,D0.W)
020428  4E75                        RTS
02042A
02042A                      * == SIEGMELDUNGEN == *
02042A
02042A  0C06 0010           OBSIEG: CMPI.B  #16,D6
02042E  6708                        BEQ.S   SIEGM
```

```
020430  0C06 0011              CMPI.B  #17,D6
020434  6708                  BEQ.S   SIEGC
020436  4E75                  RTS
020438  41FA 0056      SIEGM:  LEA     TSIEGM(PC),A0
02043C  6008                  BRA.S   SIEGUT
02043E  6100 008C      SIEGC:  BSR     ZEISIG
020442  41FA 0060              LEA     TSIEGC(PC),A0
020446  7032            SIEGUT: MOVEQ   #$32,D0
020448  720A                  MOVEQ   #$0A,D1
02044A  7428                  MOVEQ   #$28,D2
02044C  4EB9 00001386         JSR     @WRITE
020452  7021                  MOVEQ   #$21,D0
020454  7202                  MOVEQ   #2,D1
020456  7460                  MOVEQ   #$60,D2
020458  41FA 005C              LEA     TNEUSP(PC),A0
02045C  4EB9 00001386         JSR     @WRITE
020462  7021                  MOVEQ   #$21,D0
020464  323C 0098              MOVE.W  #$98,D1
020468  7460                  MOVEQ   #$60,D2
02046A  7604                  MOVEQ   #4,D3
02046C  41FA FE74              LEA     EBER(PC),A0
020470  4EB9 00001486         JSR     @READ
020476  41FA FE6A              LEA     EBER(PC),A0
02047A  0C10 004A              CMPI.B  #'J',(A0)
02047E  6706                  BEQ.S   RESTRT
020480  4FEF 0004              LEA     4(A7),A7
020484  4E75                  RTS
020486  2E47            RESTRT: MOVEA.L D7,A7
020488  4FEF 0004              LEA     4(A7),A7
02048C  6000 FB72              BRA     START
020490
020490  53696520686162 TSIEGM: DC.B    'Sie haben '
020497  656E20
02049A  6765776F6E6E65         DC.B    'gewonnen!',0
0204A1  6E2100
0204A4  496368206861 62 TSIEGC: DC.B    'Ich habe '
0204AB  6520
0204AD  6765776F6E6E65         DC.B    'gewonnen',0
0204B4  6E00
0204B6  4E6575657320   TNEUSP: DC.B    'Neues '
0204BC  537069656C3F00         DC.B    'Spiel?',0,0
0204C3  00
0204C4
0204C4  00000000       SIGFLD: DC.L    0
0204C8  00000000               DC.L    0
0204CC
```

```
0204CC  303C 01FF       ZEISIG: MOVE.W  #$1FF,D0
0204D0  41FA FFF2               LEA     SIGFLD(PC),A0
0204D4  BA34 0001       LPZEIS: CMP.B   1(A4,D0.W),D5
0204D8  6608                    BNE.S   LZEIS2
0204DA  3200                    MOVE.W  D0,D1
0204DC  5241                    ADDQ.W  #1,D1
0204DE  E641                    ASR.W   #3,D1
0204E0  10C1                    MOVE.B  D1,(A0)+
0204E2  51C8 FFF0       LZEIS2: DBRA    D0,LPZEIS
0204E6  283A FFDC               MOVE.L  SIGFLD(PC),D4
0204EA  6100 0010               BSR     ZEIKAS
0204EE  6100 000C               BSR     ZEIKAS
0204F2  6100 0008               BSR     ZEIKAS
0204F6  6100 0004               BSR     ZEIKAS
0204FA  4E75                    RTS
0204FC  48E7 FF00       ZEIKAS: MOVEM.L D0-D7,-(A7)
020500  1A04                    MOVE.B  D4,D5
020502  3405                    MOVE.W  D5,D2
020504  0242 000C               ANDI.W  #$0C,D2
020508  E442                    ASR.W   #2,D2
02050A  C4FC 001F               MULU    #31,D2
02050E  0442 017F               SUB.W   #$17F,D2
020512  4442                    NEG.W   D2
020514  3205                    MOVE.W  D5,D1
020516  E841                    ASR.W   #4,D1
020518  0241 0003               ANDI.W  #3,D1
02051C  C2FC 0005               MULU    #5,D1
020520  0245 0003               ANDI.W  #3,D5
020524  D245                    ADD.W   D5,D1
020526  C2FC 0019               MULU    #25,D1
02052A  0641 0013               ADDI.W  #$13,D1
02052E  6100 000A               BSR     KASTN
020532  4CDF 00FF               MOVEM.L (A7)+,D0-D7
020536  E09C                    ROR.L   #8,D4
020538  4E75                    RTS
02053A  4EB9 0000108A   KASTN:  JSR     @HEBE
020540  4EB9 0000122E           JSR     @AUFXY
020546  7000                    MOVEQ   #0,D0
020548  4EB9 0000121E           JSR     @AUFK
02054E  4EB9 00001090           JSR     @SENKE
020554  6100 001E               BSR     QUASEI
020558  6100 001A               BSR     QUASEI
02055C  7001                    MOVEQ   #1,D0
02055E  4EB9 00001288           JSR     @SCHREITE
020564  6100 0030               BSR     QUAKL
020568  6100 002C               BSR     QUAKL
```

```
020568  4EB9 0000108A          JSR     @HEBE
020572  4E75                   RTS
020574  7017           QUASEI: MOVEQ   #23,D0
020576  4EB9 00001288         JSR     @SCHREITE
02057C  705A                 MOVEQ   #90,D0
02057E  4EB9 000012F6        JSR     @DREHE
020584  701C                 MOVEQ   #28,D0
020586  4EB9 00001288        JSR     @SCHREITE
02058C  705A                 MOVEQ   #90,D0
02058E  4EB9 000012F6        JSR     @DREHE
020594  4E75                 RTS
020596  7015           QUAKL:  MOVEQ   #21,D0
020598  4EB9 00001288         JSR     @SCHREITE
02059E  705A                 MOVEQ   #90,D0
0205A0  4EB9 000012F6        JSR     @DREHE
0205A6  701A                 MOVEQ   #26,D0
0205A8  4EB9 00001288        JSR     @SCHREITE
0205AE  705A                 MOVEQ   #90,D0
0205B0  4EB9 000012F6        JSR     @DREHE
0205B6  4E75                 RTS
0205B8                 * == ZUG COMPUTER == *
0205B8
0205B8  6100 001C      ZUGCOM: BSR     SPIELW
0205BC  6100 006C              BSR     MAXWRT
0205C0  6100 00A8              BSR     ZWICKM
0205C4  6100 00F4              BSR     SUCHZG
0205C8  6100 0146              BSR     MULMUL
0205CC  6100 0166              BSR     ZEIGZC
0205D0  6100 019A              BSR     SETZZC
0205D4  4E75                   RTS
0205D6
0205D6  41FA 001A      SPIELW: LEA     WERTTA(PC),A0
0205DA  43FA 0262              LEA     PROTAB(PC),A1
0205DE  704B                   MOVEQ   #75,D0
0205E0  1219           LOPSPW: MOVE.B  (A1)+,D1
0205E2  4881                   EXT.W   D1
0205E4  1430 1000              MOVE.B  0(A0,D1.W),D2
0205E8  1342 0051              MOVE.B  D2,81(A1)
0205EC  51C8 FFF2              DBRA    D0,LOPSPW
0205F0  4E75                   RTS
0205F2
0205F2  00020304        WERTTA: DC.L    $00020304
0205F6  05000100                DC.L    $05000100
0205FA  08060000                DC.L    $08060000
0205FE  01000000                DC.L    $01000000
020602  00000100                DC.L    $00000100
```

```
020606  00000000                    DC.L    $00000000
02060A  00000009                    DC.L    $00000009
02060E  00000000                    DC.L    $00000000
020612  00000000                    DC.L    $00000000
020616  00000000                    DC.L    $00000000
02061A  00000000                    DC.L    $00000000
02061E  00000000                    DC.L    $00000000
020622  00000000                    DC.L    $00000000
020626  00000000                    DC.L    $00000000
02062A
02062A  45FA 0264           MAXWRT: LEA     TABSPW(PC),A2
02062E  47FA 01C8                   LEA     SPIELF(PC),A3
020632  49FA 02AE                   LEA     MLDUFN(PC),A4
020636  703F                       MOVEQ   #63,D0
020638  3200               NXTFLD: MOVE.W  D0,D1
02063A  E741                       ASL.W   #3,D1
02063C  41F4 1000                  LEA     0(A4,D1.W),A0
020640  7400                       MOVEQ   #0,D2
020642  1618               NXTMUL: MOVE.B  (A0)+,D3
020644  6B0E                       BMI.S   EXNXML
020646  4883                       EXT.W   D3
020648  1832 3000                  MOVE.B  0(A2,D3.W),D4
02064C  B404                       CMP.B   D4,D2
02064E  6402                       BCC.S   NOTEXG
020650  1404                       MOVE.B  D4,D2
020652  60EE               NOTEXG: BRA.S   NXTMUL
020654  E902               EXNXML: ASL.B   #4,D2
020656  1833 0000                  MOVE.B  0(A3,D0.W),D4
02065A  0204 000F                  ANDI.B  #$F,D4
02065E  D802                       ADD.B   D2,D4
020660  1784 0000                  MOVE.B  D4,0(A3,D0.W)
020664  51C8 FFD2                  DBRA    D0,NXTFLD
020668  4E75                       RTS
02066A
02066A  45FA 0224           ZWICKM: LEA     TABSPW(PC),A2
02066E  47FA 0188                   LEA     SPIELF(PC),A3
020672  49FA 026E                   LEA     MLDUFN(PC),A4
020676  703F                       MOVEQ   #63,D0
020678  3200               LOPZW1: MOVE.W  D0,D1
02067A  E741                       ASL.W   #3,D1
02067C  41F4 1000                  LEA     0(A4,D1.W),A0
020680  7401                       MOVEQ   #1,D2
020682  1618               LOPZW2: MOVE.B  (A0)+,D3
020684  6B12                       BMI.S   EXLZW2
020686  4883                       EXT.W   D3
020688  1832 3000                  MOVE.B  0(A2,D3.W),D4
```

```
02068C  0C04 0005              CMPI.B  #5,D4
020690  6604                   BNE.S   NOT5
020692  C4FC 0005              MULU    #5,D2
020696  60EA           NOT5:  BRA.S   LOPZW2
020698  0C42 0019     EXLZW2: CMPI.W  #25,D2
02069C  6516                   BCS.S   NOTZWI
02069E  1833 0000              MOVE.B  0(A3,D0.W),D4
0206A2  0C04 0071              CMPI.B  #'71,D4
0206A6  640C                   BCC.S   NOTZWI
0206A8  0204 000F              ANDI.B  #'0F,D4
0206AC  0004 0070              ORI.B   #'70,D4
0206B0  1784 0000              MOVE.B  D4,0(A3,D0.W)
0206B4  51C8 FFC2     NOTZWI:  DBRA    D0,LOPZW1
0206B8  4E75                   RTS
0206BA
0206BA  6100 0012     SUCHZ6:  BSR     SUCHMX
0206BE  7002                   MOVEQ   #2,D0
0206C0  4EB9 00001 1CE         JSR     @RND
0206C6  0800 0000              BTST    #0,D0
0206CA  6624                   BNE.S   SUCHVW
0206CC  6034                   BRA.S   SUCHRW
0206CE  703F          SUCHMX:  MOVEQ   #63,D0
0206D0  7400                   MOVEQ   #0,D2
0206D2  1833 0000     LOPSMX:  MOVE.B  0(A3,D0.W),D4
0206D6  0204 000F              ANDI.B  #'0F,D4
0206DA  0C04 0001              CMPI.B  #1,D4
0206DE  660A                   BNE.S   NOTFRE
0206E0  1833 0000              MOVE.B  0(A3,D0.W),D4
0206E4  B802                   CMP.B   D2,D4
0206E6  6502                   BCS.S   NOTFRE
0206E8  1404                   MOVE.B  D4,D2
0206EA  51C8 FFE6     NOTFRE:  DBRA    D0,LOPSMX
0206EE  4E75                   RTS
0206F0
0206F0  7000          SUCHVW:  MOVEQ   #0,D0
0206F2  B433 0000     LOOPVW:  CMP.B   0(A3,D0.W),D2
0206F6  6708                   BEQ.S   EXITVW
0206F8  5240                   ADDQ.W  #1,D0
0206FA  0C40 0040              CMPI.W  #64,D0
0206FE  66F2                   BNE.S   LOOPVW
020700  4E75          EXITVW:  RTS
020702
020702  703F          SUCHRW:  MOVEQ   #63,D0
020704  B433 0000     LOOPRW:  CMP.B   0(A3,D0.W),D2
020708  6704                   BEQ.S   EXITRW
02070A  51C8 FFF8              DBRA    D0,LOOPRW
```

```
02070E  4E75              EXITRW: RTS
020710
020710  41FA FBD5        MULMUL: LEA     FELDNR(PC),A0
020714  1080                      MOVE.B  D0,(A0)
020716  41FA FCCA                LEA     FAKTOR(PC),A0
02071A  30BC 0003                MOVE.W  #3,(A0)
02071E  6100 FCC4                BSR     MULPRO
020722  47FA 00D4                LEA     SPIELF(PC),A3
020726  103A FBBF                MOVE.B  FELDNR(PC),D0
02072A  4880                     EXT.W   D0
02072C  0633 0002 0000           ADDI.B  #2,0(A3,D0.W)
020732  4E75                     RTS
020734
020734  41FA 0030        ZEIGZC: LEA     CZUGST(PC),A0
020738  E518                      ROL.B   #2,D0
02073A  6116                     BSR.S   AUFBZC
02073C  6114                     BSR.S   AUFBZC
02073E  6112                     BSR.S   AUFBZC
020740  7021                     MOVEQ   #$21,D0
020742  726E                     MOVEQ   #$6E,D1
020744  7450                     MOVEQ   #$50,D2
020746  41FA 001E                LEA     CZUGST(PC),A0
02074A  4EB9 00001386            JSR     @WRITE
020750  4E75                     RTS
020752  E518              AUFBZC: ROL.B   #2,D0
020754  1200                     MOVE.B  D0,D1
020756  0201 0003                ANDI.B  #3,D1
02075A  0601 0031                ADDI.B  #$31,D1
02075E  1081                     MOVE.B  D1,(A0)
020760  41E8 0002                LEA     2(A0),A0
020764  4E75                     RTS
020766
020766  20202020        CZUGST: DC.L    $20202020
02076A  2000                    DC.W    $2000
02076C
02076C  41FA FFF8        SETZZC: LEA     CZUGST(PC),A0
020770  1428 0002                MOVE.B  2(A0),D2
020774  0242 0007                ANDI.W  #7,D2
020778  C4FC 000F                MULU    #15,D2
02077C  0442 00D1                SUBI.W  #$D1,D2
020780  4442                     NEG.W   D2
020782  1010                     MOVE.B  (A0),D0
020784  0240 0007                ANDI.W  #7,D0
020788  C0FC 0005                MULU    #5,D0
02078C  1228 0004                MOVE.B  4(A0),D1
020790  0241 0007                ANDI.W  #7,D1
```

```
020794  D240                        ADD.W   D0,D1
020796  C2FC 0019                   MULU    #$19,D1
02079A  0441 007F                   SUBI.W  #$7F,D1
02079E  7031                        MOVEQ   #$31,D0
0207A0  41FA FC3E                   LEA     SYMCOM(PC),A0
0207A4  6100 0004                   BSR     FLACKE
0207A8  4E75                        RTS
0207AA  7607            FLACKE:     MOVEQ   #07,D3
0207AC  48E7 6080       LFLACK:     MOVEM.L D1-D2/A0,-(A7)
0207B0  7011                        MOVEQ   #$11,D0
0207B2  4EB9 00001386               JSR     @WRITE
0207B8  7001                        MOVEQ   #1,D0
0207BA  4EB9 00000E5A               JSR     @DELAY
0207C0  4CD7 0106                   MOVEM.L (A7),D1-D2/A0
0207C4  7021                        MOVEQ   #$21,D0
0207C6  4EB9 00001386               JSR     @WRITE
0207CC  7001                        MOVEQ   #1,D0
0207CE  4EB9 00000E5A               JSR     @DELAY
0207D4  4CD7 0106                   MOVEM.L (A7),D1-D2/A0
0207D8  7031                        MOVEQ   #$31,D0
0207DA  4EB9 00001386               JSR     @WRITE
0207E0  7001                        MOVEQ   #1,D0
0207E2  4EB9 00000E5A               JSR     @DELAY
0207E8  4CDF 0106                   MOVEM.L (A7)+,D1-D2/A0
0207EC  51CB FFBE                   DBRA    D3,LFLACK
0207F0  4E75                        RTS
0207F2  535049454C46                DC.B    'SPIELF'
0207F8                      SPIELF: DS.B    64
020838  50524F544142                DC.B    'PROTAB'
02083E                      PROTAB: DS.B    76
02088A  544142535057                DC.B    'TABSPW'
020890                      TABSPW: DS.B    76
0208DC  4D554C44464E                DC.B    'MULDFN'
0208E2  00102030            MLDUFN: DC.L    $00102030 * 00
0208E6  384448FF                    DC.L    $384448FF
0208EA  01102145                    DC.L    $01102145
0208EE  FFFFFFFF                    DC.L    -1
0208F2  02102246                    DC.L    $02102246
0208F6  FFFFFFFF                    DC.L    -1
0208FA  03102334                    DC.L    $03102334
0208FE  3C4749FF                    DC.L    $3C4749FF
020902  04142031                    DC.L    $04142031 * 04
020906  FFFFFFFF                    DC.L    -1
02090A  05142138                    DC.L    $05142138
02090E  FFFFFFFF                    DC.L    -1
020912  0614223C                    DC.L    $0614223C
```

```
020916  FFFFFFFF        DC.L    -1
02091A  07142335        DC.L    $07142335
02091E  FFFFFFFF        DC.L    -1
020922  08182032        DC.L    $08182032 *  08
020926  FFFFFFFF        DC.L    -1
02092A  0918213C        DC.L    $0918213C
02092E  FFFFFFFF        DC.L    -1
020932  0A182238        DC.L    $0A182238
020936  FFFFFFFF        DC.L    -1
02093A  0B182336        DC.L    $0B182336
02093E  FFFFFFFF        DC.L    -1
020942  0C1C2033        DC.L    $0C1C2033 * 0C
020946  3C404BFF        DC.L    $3C404BFF
02094A  0D1C2141        DC.L    $0D1C2141
02094E  FFFFFFFF        DC.L    -1
020952  0E1C2242        DC.L    $0E1C2242
020956  FFFFFFFF        DC.L    -1
02095A  0F1C2337        DC.L    $0F1C2337
02095E  38434AFF        DC.L    $38434AFF
020962  00112439        DC.L    $00112439 * 10
020966  FFFFFFFF        DC.L    -1
02096A  01112530        DC.L    $01112530
02096E  FFFFFFFF        DC.L    -1
020972  02112634        DC.L    $02112634
020976  FFFFFFFF        DC.L    -1
02097A  0311273D        DC.L    $0311273D
02097E  FFFFFFFF        DC.L    -1
020982  04152444        DC.L    $04152444 * 14
020986  FFFFFFFF        DC.L    -1
02098A  05152531        DC.L    $05152531
02098E  394548FF        DC.L    $394548FF
020992  06152635        DC.L    $06152635
020996  3D4649FF        DC.L    $3D4649FF
02099A  07152747        DC.L    $07152747
02099E  FFFFFFFF        DC.L    -1
0209A2  08192440        DC.L    $08192440 * 18
0209A6  FFFFFFFF        DC.L    -1
0209AA  09192532        DC.L    $09192532
0209AE  3D414BFF        DC.L    $3D414BFF
0209B2  0A192636        DC.L    $0A192636
0209B6  39424AFF        DC.L    $39424AFF
0209BA  0B192743        DC.L    $0B192743
0209BE  FFFFFFFF        DC.L    -1
0209C2  0C1D243D        DC.L    $0C1D243D * 1C
0209C6  FFFFFFFF        DC.L    -1
0209CA  0D1D2533        DC.L    $0D1D2533
```

```
0209CE  FFFFFFFF        DC.L    -1
0209D2  0E1D2637        DC.L    $0E1D2637
0209D6  FFFFFFFF        DC.L    -1
0209DA  0F1D2739        DC.L    $0F1D2739
0209DE  FFFFFFFF        DC.L    -1
0209E2  0012283A        DC.L    $0012283A * 20
0209E6  FFFFFFFF        DC.L    -1
0209EA  01122934        DC.L    $01122934
0209EE  FFFFFFFF        DC.L    -1
0209F2  02122A30        DC.L    $02122A30
0209F6  FFFFFFFF        DC.L    -1
0209FA  03122B3E        DC.L    $03122B3E
0209FE  FFFFFFFF        DC.L    -1
020A02  04162840        DC.L    $04162840 * 24
020A06  FFFFFFFF        DC.L    -1
020A0A  05162935        DC.L    $05162935
020A0E  3A414AFF        DC.L    $3A414AFF
020A12  06162A31        DC.L    $06162A31
020A16  3E424BFF        DC.L    $3E424BFF
020A1A  07162B43        DC.L    $07162B43
020A1E  FFFFFFFF        DC.L    -1
020A22  081A2844        DC.L    $081A2844 * 28
020A26  FFFFFFFF        DC.L    -1
020A2A  091A2936        DC.L    $091A2936
020A2E  3E4549FF        DC.L    $3E4549FF
020A32  0A1A2A32        DC.L    $0A1A2A32
020A36  3A4648FF        DC.L    $3A4648FF
020A3A  0B1A2B47        DC.L    $0B1A2B47
020A3E  FFFFFFFF        DC.L    -1
020A42  0C1E283E        DC.L    $0C1E283E * 2C
020A46  FFFFFFFF        DC.L    -1
020A4A  0D1E2937        DC.L    $0D1E2937
020A4E  FFFFFFFF        DC.L    -1
020A52  0E1E2A33        DC.L    $0E1E2A33
020A56  FFFFFFFF        DC.L    -1
020A5A  0F1E2B3A        DC.L    $0F1E2B3A
020A5E  FFFFFFFF        DC.L    -1
020A62  00132C34        DC.L    $00132C34 * 30
020A66  3B404AFF        DC.L    $3B404AFF
020A6A  01132D41        DC.L    $01132D41
020A6E  FFFFFFFF        DC.L    -1
020A72  02132E42        DC.L    $02132E42
020A76  FFFFFFFF        DC.L    -1
020A7A  03132F30        DC.L    $03132F30
020A7E  3F434BFF        DC.L    $3F434BFF
020A82  04172C35        DC.L    $04172C35 * 34
```

```
020AB6  FFFFFFFF          DC.L    -1
020A8A  05172D3B          DC.L    $05172D3B
020A8E  FFFFFFFF          DC.L    -1
020A92  06172E3F          DC.L    $06172E3F
020A96  FFFFFFFF          DC.L    -1
020A9A  07172F31          DC.L    $07172F31
020A9E  FFFFFFFF          DC.L    -1
020AA2  081B2C36          DC.L    $081B2C36 * 38
020AA6  FFFFFFFF          DC.L    -1
020AAA  091B2D3F          DC.L    $091B2D3F
020AAE  FFFFFFFF          DC.L    -1
020AB2  0A1B2E3B          DC.L    $0A1B2E3B
020AB6  FFFFFFFF          DC.L    -1
020ABA  0B1B2F32          DC.L    $0B1B2F32
020ABE  FFFFFFFF          DC.L    -1
020AC2  0C1F2C37          DC.L    $0C1F2C37 * 3C
020AC6  3F4449FF          DC.L    $3F4449FF
020ACA  0D1F2D45          DC.L    $0D1F2D45
020ACE  FFFFFFFF          DC.L    -1
020AD2  0E1F2E46          DC.L    $0E1F2E46
020AD6  FFFFFFFF          DC.L    -1
020ADA  0F1F2F33          DC.L    $0F1F2F33
020ADE  3B4748FF          DC.L    $3B4748FF
020AE2
020AE2                    END
```

40
Zum Ausklang

Hoffentlich konnte hinreichend deutlich gemacht werden, wie effektiv sich der M68000 schon auf der User-Ebene programmieren läßt. Neben komfortablen Befehlen wie denen für Multiplikation und Division sowie Einzelbit-Zugriff tragen dazu die hervorragend gut zusammengestellten Adressierungsarten bei, die ohne Zusatzaufwand die Erstellung relokatibler Programme erlauben. Hinzu kommt der schon fast üppig zu nennende adressierbare Speicherraum von bis zu 16 MB.

Für einen so potenten Prozessor lassen sich aber auch komfortable höhere Sprachen schreiben; sowohl das SuperBASIC des QL als auch das Amiga-BASIC erfreuten durch ihre speziellen Möglichkeiten. Problematischer wird's bei den Betriebssystemen, vor allem, weil die Dokumentation nicht immer ausreicht bzw. im Einzelfall überhaupt nicht, erst spät oder nur zu hohen Preisen erhältlich ist. Üppige Auswüchse zeigen jene Computer, die Kombinationstaschenmessern (mit Nagelfeile und Motorsäge) gleichen. Der Verfasser sucht seit drei Jahren ein schlichtes, aber leistungsfähiges Datenverarbeitungsgerät mit Prozessor M68000. Man könnte damit Organisationsprobleme lösen, die in der Bundesrepublik zehntausendfach immer wieder von Hand gelöst werden. Was nützen dabei eine «Benutzeroberfläche», ein MIDI-Anschluß, hochauflösende Grafik oder ein konzertreifer Tongenerator? Müssen sich EDV-Anwender, die ohne Klimbim mit dem M68000 *e*lektronisch *D*aten *v*erarbeiten wollen, ihre Computer selber bauen?

Anhang

A) Literaturverzeichnis

a) Allgemeines/Grundlagen zum Thema

CZERWINSKI, M.: *Testen Sie Ihr Mikrowissen,* Band 2. Würzburg: Vogel-Buchverlag, 1985.
ROLLE, G.: *Mikrowissen.* Braunschweig/Wiesbaden: Vieweg, 1985.

b) M68000-Familie

FLOEGEL, E.: *68000 Programmierhandbuch.* Holzkirchen: Hofacker, 1984.
HILF, W., und NAUSCH, A.: *M68000-Familie,* Teil 1. München: te-wi, 1984.
HILF, W., und NAUSCH, A.: *M68000-Familie,* Teil 2. München: te-wi, 1985.
KOCH, J. (Valvo): *Der 16-bit-Mikroprozessor SC 68000, Befehlsvorrat.* Hamburg: Boysen & Maasch, 1983.
KOCH, J. (Valvo): *Der 16-bit-Mikroprozessor SC 68000, Eigenschaften.* Hamburg: Boysen & Maasch, 1983.
KOLLER, JH.: *16-bit-Mikrocomputer.* Holzkirchen: Hofacker, 1981.
MOTOROLA: *MC68000 16-bit-Mikroprozessor-Benützer-Handbuch.* 2. dt. Aufl.
VIEILLEFOND, C.: *Programmierung des 68000.* Düsseldorf: Sybex, 1985.

c) Bücher zum Amiga

BREUER, M.: *Das Amiga-Handbuch.*
HAAR: *Markt & Technik,* 1986.
HESSELMANN, N.: *Der Amiga.* Düsseldorf: Sybex, 1985.
SPANIK, CH.: *Amiga für Einsteiger.* Düsseldorf: Data Becker, 1986.

d) Bücher zum NDR-Computer

KLEIN, R.-D.: *Das 68000/8-Grundprogramm.* München: Franzis'-Software Service, 1985.

KLEIN, R.-D.: *Die Prozessoren 68000 und 68008.* München: Franzis', 1986.
NN: *Handbuch zum Systema-DOS.* Systema-Soft, Eichenstr. 2c, 4920 Lemgo 1

e) Bücher zum Sinclair QL

DICKENS, A.: *QL Advanced User Guide.* Cambridge (GB): Adder Publishing, 1984. (ü)
LAUFER, M., und LUFT, R.: *Der Sinclair QL unter Kontrolle.* Giga Soft/ABC Elektronic, Bielefeld, 1985. (ü)
Metacomco Assembler Manual
NAYLOR, J., and ROGERS, D.: *Inside the Sinclair QL, An introductory guide to the hardware.* London: Sunshine Books, 1985.
OPIE, C.: *QL Assembly Language Programming.* Maidenhead: McGraw-Hill, 1984. (ü)
PENNELL, A.: *The Sinclair QDOS Companion, A guide to the QL operating system.* London: Sunshine Books. (ü)

(ü) = einige inhaltliche Überschneidungen.

f) Zeitschriften

OSTERMANN, H.: *Assemblerlehrgang 68000. CHIP Mikrocomputer-Magazin.* Würzburg: Vogel, ab 2/86.

B) Programmierung eines Mikroprozessors

Für die mit den Grundlagen des Programmierens in der Maschinensprache weniger vertrauten Leser sei der zugrundeliegende Sachverhalt hier in geschlossener Form kurz dargestellt. Einzelne Fachausdrücke wird man dazu im Glossar oder in der Fachliteratur näher erläutert finden.

Zunächst einmal ist ein Mikroprozessor ein hochkomplexer elektronischer Baustein, der äußerst schnell nach Aufnahme elektrisch codierter Zahlen (s. Bitmuster) das Ergebnis der internen Verarbeitung ebenfalls in Form elektrisch codierter Zahlen ausgibt.

Nur am Rand sei erwähnt, daß zum Betrieb eines Prozessors eine Stromversorgung und ein Taktgenerator (z.B. 8 MHz) nötig sind. Unter dem Gesichtspunkt des Programmierens sind zwei andere Baugruppen wichtig:

a) ROM oder EPROM, ein Nurlese-Baustein (read only memory), in den vor der Verwendung Bitmuster eingebrannt werden. In diesem Speicherbaustein findet der Prozessor neben anderen zwei unerläßliche Angaben: Wo findet er die Anweisungen, die er abarbeiten soll (also das Programm)? Wo kann er seine Zwischennotizen ablegen (Stapel)?
b) In aller Regel wird auch noch ein RAM benötigt, ein Schreib-/Lesespeicher mit willkürlichem Zugriff (random access memory). Seine beiden Hauptaufgaben sind: Aufnahme und Wiedergabe der Zwischennotizen des Prozessors (Stapeleintragungen) und Aufnahme und Wiedergabe des Maschinenprogramms, einer sinnvollen Folge wohldefinierter Bitmuster.

Wegen der hohen Verarbeitungsgeschwindigkeit können dem Prozessor die Befehle nicht in einem Direktmodus zugeführt werden. (Welcher Mensch könnte wohl in einer Sekunde vier Millionen Ziffern eintippen?) Die einzelnen Befehle (aktive Daten) und die zu verarbeitenden (passiven) Daten werden statt dessen in einem schnellen elektronischen Medium, dem Speicher, bereitgestellt.

Das Grund-Datenformat der 8-bit-Prozessoren (s. Bit) ist das 8 bit breite Byte (s.d.). Der M68000 ist ein 16-bit-Prozessor; sein Grund-Datenformat ist das 16 bit breite Wort. Ein heute für 7 DM erhältlicher Speicherbaustein kann z.B. 8 × 1024 Byte = 8 KB bzw. 4 × 1024 (Daten-)Wörter aufnehmen, wobei für den Speicher kein Unterschied zwischen aktiven und passiven Daten besteht.

Welcher Speicher, RAM oder (EP)ROM, angemessen ist, hängt auch von der Aufgabe des Computers ab. Wenn er immer die gleiche Aufgabe nach

C) Lösungen und Lösungsvorschläge zu den Aufgaben

L.3.1:

a) $02468A − $023456 = $1234. Es ist stellenweise zu rechnen: 10 − 6 = 4 usw.

b) $01EFEC − $01DBCA = $1332. 12 − 10 = 2; 14 − 11 = 3 usw.

c) $035370 − $0350CE = $02A2. Hier ist mit 16er-Überträgen zu rechnen: 0 − 14 «geht nicht». Einen 16er borgen: 16 − 14 = 2. 6 − 12 «geht nicht»; einen 16er borgen: 16 + 6 = 22; 22 − 12 = 10, also A. Usw.

d) Vorsicht Glatteis! $01D234 − $015234 = $008000. Die hier zur Debatte stehenden Adreßdistanzen sind aber vorzeichenbehaftet auf 16 bit begrenzt, und $015234 + $8000 = $00D234, da $8000 hier als negative Zahl gilt, denn das höchste Bit ist gesetzt. Mit anderen Worten: Gefragt war nach einer Vorwärtsdistanz (in Programmablaufrichtung); die Distanz $8000 beschreibt aber einen Rückwärtssprung! Hier wurde eine der beiden Grenzen einer 16-bit-Distanz überschritten (die zulässige Vorwärtsdistanz) und die andere erreicht; $8000 repräsentiert die weiteste zulässige Rückwärtsdistanz.

e) Zugrunde zu legen ist PC = 015236. $01D234 − $015236 = $7FFE. Diese Distanz ist einen Schritt kürzer als das zulässige Maximum.

L.3.2:

a) Adreßregister indirekt.

b) 0000003B.

c1) 70A3.

c2) FFFFFFA3: Der MOVEQ-Befehl erweitert die 8-bit-Konstante vorzeichenrichtig auf .L-Format.

L.4.1:

	009C0E	9C10	9C12	9C14
Nach der 1. Ablage:	FFFF	FFAB		
Nach der 2. Ablage:	FFFF	FFAB	FFAB	
Nach der 3. Ablage:	FFFF		FFAB	AB

L.4.2a: Mehrfache Ablage mit dem QL

```
03F800  70AB       MOVEQ    #$AB,D0
03F802  41FA 000C  LEA      $000C(PC),A0
03F806  20C0       MOVE.L   D0,(A0)+
03F808  30C0       MOVE.W   D0,(A0)+
03F80A  10C0       MOVE.B   D0,(A0)+
03F80C  7000       MOVEQ    #0,D0
03F80E  4E75       RTS
```

gleichen Regeln zu lösen hat (z.B. Heizungsregelung), kann man das Programm in einen Festwertspeicher (EPROM) einbrennen. Soll der Rechner aber mal dieses und mal jenes tun, muß er vor Arbeitsbeginn programmiert werden, dann wird das Programm in ein RAM eingegeben.

Der hier gemeinte Speicherbaustein (6264) enthält 8192 Byte-Speicherstellen, die zunächst einmal von 0 bis 8191 bzw. $1FFF numeriert sind. Durch entsprechende Schaltung läßt sich aber derselbe Baustein auch auf die Adressen von $2000 bis $3FFF oder von $4000 bis $5FFF usw. legen. Mit entsprechend vielen Speicherbausteinen kann man so für den M68000 einen von $000000 bis $FFFFFF reichenden Speicher bauen. Die Nummern der Speicherstellen nennt man Adressen. Durch Vorgabe einer bestimmten Adresse kann man ein Datum (Einzahl von «Daten») in einer ganz bestimmten Speicherstelle ablegen bzw. aus ihr auslesen. Die Verbindung zu den einzelnen Speicherstellen wird dadurch hergestellt, daß man das der gewünschten Adresse entsprechende Bitmuster auf den «Adreßbus» legt: Ein Bus ist ein Bündel paralleler Leitungen, auf der Platine realisiert in Form von kupfernen Leiterbahnen. Drei solcher Busse besorgen im Mikrocomputer den Datentransfer:

Das auf dem Adreßbus anliegende Bitmuster aktiviert eine bestimmte Speicherstelle.

Der Datenbus überträgt die Bitmuster zwischen Prozessor, Tastatur, Speicher, Datensichtgerät, Diskettenlaufwerken, Drucker usw. (Die gemeinte Baugruppe wird ebenfalls über den Adreßbus angewählt.)

Ob nun gelesen oder geschrieben werden soll, und vieles andere mehr, bestimmt das Bitmuster auf dem Steuerbus.

Hiermit sind die Aufgaben der einzelnen Baugruppen wohl so weit beschrieben, daß auch der Newcomer dem im Hauptteil Dargestellten folgen kann.

c) Das Datenlangwort 48E7FFFE wird von 035C00 nach 035C24 kopiert. Anschließend werden A0 und A1 postinkrementiert auf 035C04 bzw. 035C28.

d) Bewegt mit entsprechendem Postinkrement das Datenlangwort 43FA000E von 035C04... nach 035C18... .

e) Der Opcode eines jeden MOVE.L ist 2 Bytes lang, transferiert aber 4 Bytes; summa summarum zwei für sich selbst und zwei für das übrige Programm. Da das «übrige Programm» 9 Wörter lang ist, muß MOVE.L hier neunmal verwendet werden.

f) Nur in der Unterbringung im Speicher (location).

g) Das Programm enthält keine absoluten Adressen; es ist relokatibel.

h) 035C24.

i) Ja.

j) Es kopiert sich selbst.

k) Ja.

l) Solange noch für diesen Zweck Speicherraum verfügbar ist.

L.4.7:

Ein sich selbst kopierendes Programm

a) für den QL,

```
03F800   41FA FFFE   LEA      $FFFE(PC),A0
03F804   43FA 0012   LEA      $0012(PC),A1
03F808   22D8        MOVE.L   (A0)+,(A1)+
03F80A   22D8        MOVE.L   (A0)+,(A1)+
03F80C   22D8        MOVE.L   (A0)+,(A1)+
03F80E   22D8        MOVE.L   (A0)+,(A1)+
03F810   22D8        MOVE.L   (A0)+,(A1)+
03F812   22D8        MOVE.L   (A0)+,(A1)+
03F814   7000        MOVEQ    #0,D0
03F816   4E75        RTS
```

b) für den NDR-Computer.

```
009C00   41FA FFFE   LEA      $FFFE(PC),A0
009C04   43FA 000E   LEA      $000E(PC),A1
009C08   22D8        MOVE.L   (A0)+,(A1)+
009C0A   22D8        MOVE.L   (A0)+,(A1)+
009C0C   22D8        MOVE.L   (A0)+,(A1)+
009C0E   22D8        MOVE.L   (A0)+,(A1)+
009C10   22D8        MOVE.L   (A0)+,(A1)+
009C12   4E75        RTS
```

L.4.2b: Mehrfache Ablage mit dem Amiga

```
035C00  48E7 8080   MOVEM.L D0/A0,-(A7)
035C04  70AB        MOVEQ    #$AB,D0
035C06  41FA 000E   LEA      $000E(PC),A0
035C0A  20C0        MOVE.L   D0,(A0)+
035C0C  30C0        MOVE.W   D0,(A0)+
035C0E  10C0        MOVE.B   D0,(A0)+
035C10  4CDF 0101   MOVEM.L  (A7)+,D0/A0
035C14  4E75        RTS
```

L.4.3: Ein zünftiger Systemabsturz ist die Folge. Der erste (.B-)Ablagebefehl setzt A0 auf eine ungerade Adresse. Dann wird versucht, dort ein Wort abzulegen; das ist unzulässig. Merken Sie sich die Faustformel: Der M68000 kennt (fast) nur gerade Adressen; ungerade sind die Ausnahme.

L.4.4a:

```
009C00 70AB         MOVEQ    #$AB,D0
009C02 41FA FFFC    LEA      $FFFC(PC),A0
009C06 20C0         MOVE.L   D0,(A0)
009C08 30C0         MOVE.W   D0,(A0)
009C0A 10C0         MOVE.B   D0,(A0)
009C0C 4E75         RTS
```

L.4.4b: 009C0E: FFFF FFAB FFAB ABC0 30C0 ...

c) Beliebig oft, denn der Befehl ist selbst 2 Bytes lang, überschreibt aber nur eins; der gültige Programmrest würde also mit jedem zusätzlichen Befehl länger werden.

d) Der 2-Byte-Befehl überschreibt 2 Bytes.

e) Genau dreimal. Das nächste MOVE.L würde den Opcode 4E75 tilgen.

L.4.5:

a) 009C10: 4E75 4E78 5678 1234 5678.

b) Durch Predekrement wird A0 von 009C1A auf 009C16 vermindert. Erster Eintrag in 009C16 ... 9.

c) 009C13.

d) Das zweite RTS sollte lediglich eine Orientierungshilfe sein. Beim dritten Transfer wird 75 überschrieben.

L.4.6:

a) A0 = 035C06 + FFFFFA = 035C00.

b) 035C0A + 001A = 035C24.

b)

```
03FB00   48E7 FFFE            MOVEM.L D0-D7/A0-A6,-(A7)
03FB04   4DFA 0014            LEA     BUFFER(PC),A6
03FB08   7A19                 MOVEQ   #26-1,D5
03FB0A   1CEE FFE6  LOOP:     MOVE.B  -26(A6),(A6)+
03FB0E   51CD FFFA            DBRA    D5,LOOP
03FB12   4CDF 7FFF            MOVEM.L (A7)+,D0-D7/A0-A6
03FB16   7000                 MOVEQ   #0,D0
03FB18   4E75                 RTS
03FB1A             BUFFER:    DS.B    26
```

L.6.2:

a) 100

b) 13

c) 009C1A

d) Ersetzen Sie in 009C04 ... 0016, und schieben Sie nach DBEQ ein in 009C26 32C0 MOVE.W D0,(A1)+.

L.8.1:

```
040000   6004          LABEL1:  BRA.S   LABEL3
040002   6200 0004      LABEL2:  BHI     LABEL4
040006   62F8          LABEL3:  BHI.S   LABEL1
040008   5DCD FFF6      LABEL4:  DBLT    D5,LABEL1
04000C   66FE          LABEL5:  BNE.S   LABEL5
04000E   6000 FFF2      LABEL6:  BRA     LABEL2
```

L.10.1: Das würde zu dem in Abschnitt 8.1 erklärten Widerspruch führen.

L.15.1: 1 + 11 + 111 + 1111 + 11111 ... + $7FFFFFFF = $FFFFFFDF, wie man leicht nachrechnen lassen kann:

```
035C00   48E7 FFFE            MOVEM.L D0-D7/A0-A6,-(A7)
035C04   7000                 MOVEQ   #0,D0
035C06   223C 7FFF
         FFFF                 MOVE.L  #$7FFFFFFF,D1
035C0C   741E                 MOVEQ   #30,D2
035C0E   D081          LOOP:  ADD.L   D1,D0
035C10   E289                 LSR.L   #1,D1
035C12   51CA FFFA            DBRA    D2,LOOP
035C16   41FA 000A            LEA     BUFER(PC),A0
035C1A   2080                 MOVE.L  D0,(A0)
035C1C   4CDF 7FFF            MOVEM.L (A7)+,D0-D7/A0-A6
035C20   4E75                 RTS
035C26             BUFER:     DS.L    1
```

L.5.1:

a) 1401.

b) Hier ist An als Ziel nicht zugelassen (siehe Bild 5.1).

c) 260C.

d) 1C85.

e) Die Codierung der Adressierungsart PC-relativ wurde noch nicht besprochen. Vielleicht aber haben Sie sich an die relokatiblen Programme P.3.x erinnert und damit die Informationslücke überbrückt: 10BA 0023.

f) 3E3A FF82.

g) Hier ist zweierlei verboten: d(PC) ist als Ziel nicht zugelassen; es wird versucht, ein Langwort (ebenso für Wort) in einer ungeraden Adresse abzulegen. (Der Inhalt von PC ist immer geradzahlig.)

L.5.2:

a) 7012

b) 7234

c) 7438

d) 76B4

e) 788A

f) 7A9A

g) Hier wurde der zulässige Wertebereich überschritten.

Betthupferl: Wieso rutschte in d) die Konstante −76 ins MSB des Opcodes?

L.5.3:

a)

```
  0003F856
+ 0005CDEE
+       10
----------
= 0009C654
```

b)

```
  0003F856
+ 0005CDEE
+ FFFFFF9C
----------
= 0009C5E0
```

c)

```
  0003F856
+ FFFFCDEE
+       23
----------
= 0003C667
```

d)

```
  0005CDEE
+ 456789AB
+       67
----------
= 456D5800
```

e)

```
  0005CDEE
+ FFFF89AB
+       67
----------
= 00055800
```

f)

```
  0005CDEE
+ FFFF89AB
+ FFFFFFAB
----------
= 00055744
```

L.5.4: Sehen Sie in P.5.1 nach!

L.6.1:

a) Quelle: Adreßregister indirekt mit Adreßdistanz: Ziel: Adreßregister indirekt mit Postinkrement.

```
020000E  D308          LPADDX:  ADDX.B   -(A0),-(A1)
020010   51C8 FFFC               DBRA     D0,LPADDX
020014
020014   44FC 0000               MOVE     #0,CCR
020018   7007                   MOVEQ    #7,D0
02001A   C308          LPABCD:  ABCD     -(A0),-(A1)
02001C   51C8 FFFC               DBRA     D0,LPABCD
020020
020020   4E75                   RTS
020022                         DS.B     14
020030   12345678      BUFFER:  DC.L     $12345678
020034   23456789               DC.L     $23456789
020038   12345678               DC.L     $12345678
02003C   23456789               DC.L     $23456789
020040   55555555               DC.L     $55555555
020044   55555555               DC.L     $55555555
020048   55555555               DC.L     $55555555
02004C   55555555               DC.L     $55555555
020050   55555555               DC.L     $55555555
020054   55555555               DC.L     $55555555
020058   55555555               DC.L     $55555555
02005C   55555555               DC.L     $55555555
020060
020060                         END
```

L.18.3: ABCD mit Hexziffern

```
020000
= 00000023             DEZZAL.  EQU      $23
020000
020000   41FA 003E              LEA      REFE1(PC),A0
020004   700F                   MOVEQ    #$F,D0
020006   10FC 0023     LOOP1:   MOVE.B   #DEZZAL,(A0)+
02000A   51C8 FFFA               DBRA     D0,LOOP1
02000E
02000E   700F                   MOVEQ    #$F,D0
020010   10C0           LOOP2:   MOVE.B   D0,(A0)+
020012   51C8 FFFC               DBRA     D0,LOOP2
020016
020016   41FA 0039              LEA      REFE2+1(PC),A0
02001A   700F                   MOVEQ    #$F,D0
02001C   1220           LOOP3:   MOVE.B   -(A0),D1
02001E   1428 0010               MOVE.B   16(A0),D2
020022   44FC 0000               MOVE     #0,CCR
020026   C501                   ABCD     D1,D2
020028   1142 0020               MOVE.B   D2,32(A0)
```

L.16.1: Wenn man beide Zahlen mit EOR verknüpft, wird das höchstwertige Bit nur dann gesetzt, wenn nur eine von ihnen negativ ist.

L.16.2: Probieren Sie es doch aus. Wegen $EDCBA988 + $12345678 = (1)00000000 erkennen Sie, daß alles seine Richtigkeit hat:
NEG.L($00000000) = $00000000, NEG($80000000) = $80000000.

```
020000  41FA 000C              LEA     BUFFER(PC),A0
020004  4498                  NEG.L   (A0)+
020006  4498                  NEG.L   (A0)+
020008  4498                  NEG.L   (A0)+
02000A  7000                  MOVEQ   #0,D0
02000C  4E75                  RTS
02000E
02000E  00000000        BUFFER: DC.L    0
020012  80000000              DC.L    $80000000
020016  12345678              DC.L    $12345678
02001A
02001A                        END
```

L.18.1:

a) Die Codierung von D U E N G E R
ist 44 55 45 4E 47 45 52.
Nach Ausmaskieren bleibt 4 5 5 14 7 5 2.
4564752 = $45A710.

b) $1BCC8 = 1 1 3 8 6 4. Aus den Möglichkeiten
A A C H F D
oder 5 14
E N
oder Q Q S X V T usw. braucht man nur noch die richtige herauszufinden.
Weitere Deutungsmöglichkeiten sind z.B. ...64 = ...50 + ...14 mit Decodierung E N oder U N usw.

L.18.2: ADD und ABCD

```
020000  41FA 003E              LEA   BUFFER+16(PC),A0
020004  43FA 005A              LEA   BUFFER+48(PC),A1
020008
020008  44FC 0000              MOVE    #0,CCR
02000C  7007                  MOVEQ   #7,D0
```

L.21.5:

a) 0201BC LEA 4(A7),A7 erhöht den Stackpointer um 4, verwirft also eine Adresse;

b) Rückkehr also statt nach 0200BA nach 0200BE BRA MENUE.

L.21.6:

a) Einmal nach COMPA.

b) Wie unser Bild zeigt, dreimal nach SORTI2 und einmal zu der Ebene, die SORTIR aufrief, also zum Menü.

```
SORTIR:  BSR EINGAB      EINGAB:  ... .....
                         EINGEB:  ... .....   (1. Wort)
                                  RTS
SORI2:   BSR VERGL       VERGL:   ... .....
                         AUFSCH:  ... .....
                         EINSCH:  ... .....
                                  BRA EINGEB
                         EINGEB:  ... .....   (2. Wort)
                                  RTS
         BRA SORTI2
SORTI2:  BSR VERGL       VERGL:   ... .....
                         AUFSCH:  ... .....
                         EINSCH:  ... .....
                                  BRA EINGEB
                         EINGEB:  ... .....   (3. Wort)
                                  RTS
         BRA SORTI2
SORTI2:  BSR VERGL       VERGL:   ... .....
                         AUFSCH:  ... .....
                         EINSCH:  ... .....
                                  BRA EINGEB
                         EINGEB:  ... ....      (cr)
                                  LEA 4(A7),A7
                                  RTS
```

L.21.6 Programmablauf bei A.21.6

L.21.7: Ein solches Programm könnte so aussehen:

```
020000
020000  47FA 001C          LEA     BUFFER(PC),A3
020004
020004  7E08                MOVEQ   #8,D7
020006  4E42                TRAP    #2
020008  26C8                MOVE.L  A0,(A3)+
```

```
020020  51C8 FFEE                    DBRA    D0,LOOP3
020030  4E75                         RTS
020032
020032                       LEER:   DS.B    14
020040                       REFE1:  DS.B    16
020050                       REFE2:  DS.B    16
020060                       SUMME:  DS.B    16
020070
020070                               END
```

Sie erkennen ohne Mühe:

23	23	23
+0F	+0E	+0D
=xx	=37	=36

xx: Hier ergibt sich mal $36, aber auch $4E.

L.20.1: Sind das neue Wort und der Vergleichsrecord einander gleich, wird die Vergleichsschleife nicht vorzeitig unterbrochen, sondern der Scheifenzähler wird ganz heruntergezählt. Danach wird der Recordpointer A1 um 34 erhöht. Das neue Wort wird also hinter dem (gleichen) alten Record eingefügt.

L.20.2:

a) Nein, denn A2 wird noch vor der Verschiebung dekrementiert. Verschoben wird also von $A2_{neu}$ nach $A2_{neu} + 34$.

b) Nein, denn bei Verschiebung eines längeren Speicherbereichs würde dieser sich mit sich selbst überschreiben, und das mehrmals.

L.21.1: Bei genauer Auslegung der Frage nur einmal, in 020174 mit ADDQ.L #4,(A7).

L.21.2: $4 * (50 + 1) = 204$ Bytes.

L.21.3: A2 weist nicht auf Anfang PUDDING, sondern auf die Endnull von GOCKEL. Wir tun des Guten zuviel, indem wir die Schlußnull von BAD auch noch mit der Schlußnull von GOCKEL überschreiben. So werden zwei Befehle gespart: Die Erhöhung von A2 um 1 und die Subtraktion D0 − 1.

L.21.4:

a) Hier wird die Rücksprungadresse 0200BA im Stapel auf 0200BE erhöht (vgl. L.21.1).

b) «M».

c) ... AND 8 THEN z$="n"
d) ... AND 16 THEN z$="x"

L.23.2: ADD.B D1,D0 mit Opcode D001
wird zu SUB.B D1,D0 mit Opcode 9001
bzw. zu SUB.B D0,D1 mit Opcode 9200.

L.26.1:

a) Es handelt sich um das MSW aus D0. Da dieses von der Multiplikation nicht verändert wird, muß der zufällige Wert schon vorher dort vorhanden gewesen sein.
b) Laden Sie D0 und D1 zu Anfang willkürlich mit markanten Zahlen.

L.26.2: Ein derartiges Programm könnte so aussehen:

```
020000
020000  41FA 003E                LEA     ZALSTA(PC),A0
020004  3228 0002                MOVE.W  02(A0),D1
020008  C2E8 0008                MULU    08(A0),D1
02000C  3428 0004                MOVE.W  04(A0),D2
020010  C4E8 000A                MULU    10(A0),D2
020014  3628 0004                MOVE.W  04(A0),D3
020018  C6E8 0008                MULU    08(A0),D3
02001C  6114                     BSR.S   ADDSWA
02001E  3628 0002                MOVE.W  02(A0),D3
020022  C6E8 000A                MULU    10(A0),D3
020026  610A                     BSR.S   ADDSWA
020028  2141 000E                MOVE.L  D1,14(A0)
02002C  2142 0012                MOVE.L  D2,18(A0)
020030  4E75                     RTS
020032
020032  4842            ADDSWA:  SWAP    D2
020034  D443                     ADD.W   D3,D2
020036  4842                     SWAP    D2
020038  4843                     SWAP    D3
02003A  48C3                     EXT.L   D3
02003C  D383                     ADDX.L  D3,D1
02003E  4E75                     RTS
020040
020040                  ZALSTA:  DS.W    1
020042  1234ABCD                 DC.L    $1234ABCD
020046                           DS.W    1
020048  567890EF                 DC.L    $567890EF
02004C                           DS.L    3
020058
020058                           END
```

```
02000A  26D0                    MOVE.L   (A0),(A3)+
02000C
02000C  7E06                    MOVEQ    #6,D7
02000E  4E42                    TRAP     #2
020010  26C8                    MOVE.L   A0,(A3)+
020012  26D0                    MOVE.L   (A0),(A3)+
020014
020014  7E07                    MOVEQ    #7,D7
020016  4E42                    TRAP     #2
020018  26C8                    MOVE.L   A0,(A3)+
02001A  26D0                    MOVE.L   (A0),(A3)+
02001C
02001C  4E75                    RTS
02001E              BUFFER:     DS.L     6
020036                          END
```

Hier werden gleich die Adressen dreier Speicherfelder und die in ihnen stehenden Daten abgefragt.
Ergebnis:

Ladestartmerker:	00C788	0001 0400
Textanfangmerker:	008036	0001 0000
Textendemerker:	00803E	0001 01BC

Deutung(sversuch): Nachdem der aktuelle Quelltext auf Diskette aufgezeichnet wurde, wurde der Inhalt des Ladestartmerkers wahrscheinlich auf ein volles KB erhöht, was vielleicht der Diskettenformatierung entspricht. Die Merkeradresse 00C788 liegt im Arbeitsbereich von Systema-DOS.

Textanfang- und Textendemerker liegen im Bereich der Systemvariablen des Grundprogramms. Zu ihren Inhalten: Unter Systema-DOS werden, sofern nicht anderes bestimmt wurde, Texte ab Default-Adresse $10000 im Speicher abgelegt. Das END des Quelltextes endet in $0101AB. Es schließen sich mehrere 0D 0A (CRLF) an, möglicherweise um den Seitenvorschub zu steuern. 0101BC ist die erste Adresse nach dem letzten CRLF.

L.22.1:

a) Byte, Byte, Langwort.
b) LSB des ersten Wortes: Exponent (MSB wohl systemspezifischer Offset).
c) Wenn Sie die Mantisse als Binärzahl schreiben, wird das deutlich: Lesen Sie im Text eine halbe Seite weiter ...

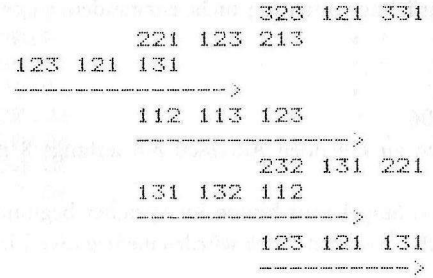
L.23.1: Ändern Sie in der inneren FOR-NEXT-Schleife:

a) ... AND 2 THEN z$="v"
b) ... AND 4 THEN z$="z"

```
022FF2  2140 0008          QUOTIF: MOVE.L  D0,8(A0)
022FF6  4E71               NULDIV: NOP
022FF8  4E71               NULQUO: NOP
022FFA  6100 010C                  BSR     DROP
022FFE  4E75                       RTS
023000
= 00023000                 EINGEB  EQU     $23000
= 0002308C                 AUSGEB  EQU     $2308C
= 00023108                 DROP    EQU     $23108
= 0002315E                 ZALSTA  EQU     $2315E
023000
023000                             END
```

L.28.1:

a)



b) 15 Züge

c) 4 * (4−1) + 1 = 13, leider! Sie können aber später mit dem Programm die richtige Formel leicht finden.

L.28.2:

a) Denken Sie daran, daß die DBRA-Schleife bei Zählerstand 0 noch durchlaufen wird!

b) Bei nur drei Turmpositionen können sich die Bewegungsüberschüsse nicht aufaddieren. Abhilfe: Das 150fache muß um 1 vermindert werden.

c) Scheiben um 2 oder 3 Pixel verschieben und dann entsprechend länger verzögern. (Beim NDR-Computer kann man außerdem den Grafikprozessor synchronisieren.)

d) Wenn VONTU + BISTU <> 4, dann MAXHO nur für zwei Türme ermitteln lassen.

L.29.1: Ziehen Sie doch den Hexdump! So kann man in Programmen unauffällig Informationen verstecken, z.B. zum Aufspüren von Raubkopie-(rer)n.

L.27.1:

a) Wenn die Division nicht ohne Rest aufgeht, wird der Dividend endlos nach links aufgeschoben und vermindert, aufgeschoben und vermindert. Das Rechenwerk läuft endlos über.

b) Sobald der Quotient bis Bit30 aufgeshiftet wurde, ist Schluß (siehe P.27.3).

L.27.2: FP-Division

a) Der Fall a, bei dem die Mantisse des Teilers größer ist als die des Dividenden, liefert ein um eine Zweierpotenz zu großes Ergebnis. Relativ zu den drei anderen (vgl. Listing) muß also der resultierende Exponent um eins vermindert werden.

b) Das fertige Programm liefert recht brauchbare Ergebnisse:

```
020000
020000  6100 2FFE        START:  BSR     EINGEB
020004  6100 2FFA                BSR     EINGEB
020008  6100 2FAA                BSR     DIVIDI
02000C  6100 307E                BSR     AUSGEB
020010  6000 FFEE                BRA     START
020014
022FB4                           ORG     $22FB4
022FB4
022FB4  41FA 01A8        DIVIDI: LEA     ZALSTA(PC),A0
022FB8  3010                     MOVE.W  (A0),D0
022FBA  9168 0006                SUB.W   D0,6(A0)
022FBE  2228 0002                MOVE.L  2(A0),D1
022FC2  6732                     BEQ.S   NULDIV
022FC4  2628 0008                MOVE.L  8(A0),D3
022FC8  672E                     BEQ.S   NULQUO
022FCA  7000                     MOVEQ   #0,D0
022FCC  B681                     CMP.L   D1,D3
022FCE  6506                     BCS.S   VERGL
022FD0  0668 0001 0006           ADDI.W  #1,6(A0)
022FD6  44FC 0010        VERGL:  MOVE    #$10,CCR
022FDA  B681                     CMP.L   D1,D3
022FDC  6B02                     BMI.S   NOTABZ
022FDE  9681                     SUB.L   D1,D3
022FE0  0A3C 0010        NOTABZ: EORI    #$10,CCR
022FE4  D180                     ADDX.L  D0,D0
022FE6  0800 001E                BTST    #30,D0
022FEA  6606                     BNE.S   QUOTIF
022FEC  D683                     ADD.L   D3,D3
022FEE  4A83                     TST.L   D3
022FF0  60E4                     BRA.S   VERGL
```

```
02002E  4E5B                        UNLK    A3
020030  4E75                        RTS
020032  4E53 0014           SUBRU3: LINK    A3,#20      * 6.
020036  6104                        BSR.S   SUBRU4      * 7.
020038  4E5B                        UNLK    A3
02003A  4E75                        RTS
02003C  4E75                SUBRU4: RTS
02003E
02003E                      STAPEL: DS.L    50
020106
020106                      * 2.  20042:  12345678
020106                      * 1.  20046:  0002001A
020106                      *     2004A:
020106                      * 4.  2004E:  00020042
020106                      * 3.  20052:  00020024
020106                      *     50056:
020106                      * 6.  2005A:  0002004E
020106                      * 5.  2005E:  0002002E
020106                      *     20062:
020106                      *     20066:
020106                      * 7.  2006A:  00020038
020106
020106                              END
```

L.30.2a Variante von P.30.2 für den NDR-Computer

Listing L.30.2a zeigt 1. absolute Adressen, 2. rechts sieben Markierungen der wesentlichen Befehle und 3. unten ein Faksimile des Stapels mit Zuordnung der Eintragungen zu den markierten Befehlen.

Wenn man nun versuchen will, die Verschachtelung mit nur einem RTS zu konstruieren, muß das UNLK offenbar in der letzten Routine stehen. Da das Programm dann aber abstürzt, zeigen wir mit der Version B einen Ausweg auf, der wenigstens zur Klärung beiträgt:

```
020000  701E                        MOVEQ   #30,D0
020002  47FA 003A                   LEA     STAPEL(PC),A3
020006  425B                LOOP:   CLR.W   (A3)+
020008  51C8 FFFC                   DBRA    D0,LOOP
02000C  284F                        MOVEA.L A7,A4
02000E  4FFA 003A                   LEA   STAPEL+12(PC),A7
020012  267C 12345678               MOVEA.L #$12345678,A3
020018  6100 0002                   BSR     SUBRU1      * 1.
02001C
02001C  4E53 0014           SUBRU1: LINK    A3,#20      * 2.
```

L.30.1:

a) Die Basisadresse $035FFC bleibt über das ganze Programm hinweg dieselbe; das gilt insbesondere für die Statements 7 und 10. In Zeile 15 wird A3 = 000006 + Basisadresse, und in Zeile 16 wird 000006 + Basisadresse + 000004 = 00000A + Basisadresse = 00000A + 035FFC = 036006.

b) Das ist der Rest der bei bsr.s xpeax abgelegten Rücksprungadresse 000008 + 035FFC = 036004; ihr MSW wurde mit AFFE überschrieben.

Die Rechnung mit relativen Adressen gewann bei der Korrektur einer Manuskriptseite eine überraschende Aktualität: Der Assembler wollte partout nicht die ursprüngliche ORG-Adresse akzeptieren. Deshalb mußten L.30.1 und L.30.2 nachträglich korrigiert werden. Beim Rechnen mit relativen Adressen wäre diese fehlerträchtige Situation nicht entstanden.

L.30.2:

a) 361CC

b) 361E6, 361F2, 361FC, 36206

c) Von den je 20 Bytes wurden zu fallenden Adressen hin anfangs 8 mit Eintragungen belegt.

d) Im allgemeinen läßt man den Stapel weit hinten im Speicher beginnen; dann kann er weit «nach vorn» wachsen. Dem würde eine negative Link-Distanz entsprechen.

e) Wenn es gar nicht klappen will, können die Gründe so verständlich gemacht werden:

```
020000  701E                        MOVEQ    #30,D0
020002  47FA 003A                   LEA      STAPEL(PC),A3
020006  425B                LOOP:   CLR.W    (A3)+
020008  51C8 FFFC                   DBRA     D0,LOOP
02000C  284F                        MOVEA.L  A7,A4
02000E  4FFA 003A                   LEA   STAPEL+12(PC),A7
020012  267C 12345678               MOVEA.L  #$12345678,A3
020018  6104                        BSR.S    SUBRU1    * 1.
02001A  2E4C                        MOVEA.L  A4,A7
02001C  4E75                        RTS
02001E
02001E  4E53 0014           SUBRU1: LINK     A3,#20    * 2.
020022  6104                        BSR.S    SUBRU2    * 3.
020024  4E5B                        UNLK     A3
020026  4E75                        RTS
020028  4E53 0014           SUBRU2: LINK     A3,#20    * 4.
02002C  6104                        BSR.S    SUBRU3    * 5.
```

L.31.2:

a) Sie haben dafür mehrere Möglichkeiten, z.B. MOVE to SR, logische Verknüpfungen mit dem SR, STOP. Da Sie diese aber (bisher) nur aus dem User-Modus heraus einsetzen können, würden Sie damit eine Privilegverletzung begehen und erhielten eine Fehlermeldung: Der QL-Monitor verliert sich im Nirwana; der NDR-Computer meldet ADRERROR; der Amiga «Privileg Violation at $035EDC».

b) Angenommen, die Interruptmaske würde durch MOVE to SR bestimmt: Dieser Befehl wird 0100 0110 11mm mrrr codiert. Ein solcher Opcode (leider aber auch Operand) ist gefunden, wenn das geprüfte Wort nach Undieren mit FFC0 46C0 ergibt. Damit ergibt sich für den NDR-Computer, dessen Grundprogramm von 0000 bis 7FFF im ROM steht, folgende Lösungsmöglichkeit:

```
000'  91C8                        SUBA.L   A0,A0
002'  43FA 001E                   LEA      BUFFER(PC),A1
006'  B1FC 0000 8000   LOOP:      CMPA.L   #$8000,A0
00C'  6712                        BEQ.S    EXIT
00E'  3018                        MOVE.W   (A0)+,D0
010'  0240 FFC0                   ANDI.W   #$FFC0,D0
014'  0C40 46C0                   CMPI.W   #$46C0,D0
018'  66EC                        BNE.S    LOOP
01A'  2288                        MOVE.L   A0,(A1)
01C'  5599                        SUBQ.L   #2,(A1)+
01E'  60E6                        BRA.S    LOOP
020'  4E75             EXIT:     RTS
022'                   BUFFER:   DS.L     50
                                 END
```

L.32.1: MOVE USP,Ax kopiert den Inhalt des USP in ein Adreßregister. MOVE.L #$123456,(Ax) überschreibt die momentane Rücksprungadresse.

Diese Befehlskombination ist nur funktionell interessant, denn das Betriebssystem «kennt» ja die Intentionen des Users nicht!

L.33.1: Offenbar war beim Eintritt in den Supervisor-Modus Z-Flag gesetzt. Da weder TRAP noch MOVE from SR noch LEA die Flags verändern, muß das Z-Flag schon beim ersten Durchlauf von LISTAP gesetzt worden sein. Da aber auch DBRA die Flags nicht beeinflußt, muß beim letzten Transfer (A4)+,(A3)+ ein Nullwort übertragen worden sein. Bild 33.1 zeigt, daß das richtig ist (3F83A). Zur Kontrolle könnten Sie aber auch vor dem Eintritt in den Supervisor-Modus das CCR willkürlich mit markanten Daten < $20 laden.

```
020020  6104                    BSR.S   SUBRU2    * 3.
020022  4E71                    NOP
020024  4E71                    NOP
020026  4E53 0014       SUBRU2: LINK    A3,#20    * 4.
02002A  6104                    BSR.S   SUBRU3    * 5.
02002C  4E71                    NOP
02002E  4E71                    NOP
020030  4E53 0014       SUBRU3: LINK    A3,#20    * 6.
020034  6104                    BSR.S   SUBRU4    * 7.
020036  4E71                    NOP
020038  4E71                    NOP
02003A  2E4C             SUBRU4: MOVEA.L A4,A7
02003C  4E75                    RTS
02003E
02003E                   STAPEL: DS.L    50
020106
020106                   * 2. 20042: 12345678
020106                   * 1. 20046: 0002001C
020106                   *    2004A:
020106                   * 4. 2004E: 00020042
020106                   * 3. 20052: 00020022
020106                   *    50056:
020106                   * 6. 2005A: 0002004E
020106                   * 5. 2005E: 0002002C
020106                   *    20062:
020106                   *    20066:
020106                   * 7. 2006A: 00020036
020106
020106                          END
```

L.30.2b Lösungsanalyse mit dem NDR-Computer

Um möglichst auf gleiche Adressen zu kommen, wurden fortgelassene Befehle durch NOPs ersetzt. Am Ende geht es völlig unkonventionell nicht ins Hauptprogramm zurück, sondern der Stackpointer wird wiederhergestellt. (Eigentlich hätten hier UNLK A3 und RTS stehen müssen.) So kehrt der Programmablauf ohne Fehlermeldung zurück, und es wird möglich, noch einmal in Ruhe in den Stapel zu schauen. Die Eintragungen entsprechen ziemlich genau der a-Version, doch wenn SUBRU4 ein UNLK enthielte, müßten im Stapel zwei gültige Adressen bereitstehen. Hier ist nur eine vorhanden; daher der unkontrollierte Ausstieg.

L.31.1: Sie brauchen dazu nur in P.23.1 «das andere Byte» mit PEEK(anfad+24) auszulesen.

A7 weist auf den TOS, der 03FADE enthält; eine Bestätigung der hohen Wahrscheinlichkeit, daß ein Job während einer Warteschleife suspendiert wird.
SR: Während der Warteschleife ist fast nie ein Flag gesetzt.
PC: Das Programm wurde tatsächlich bei LWART unterbrochen.

L.35.3: POKEn Sie nach 03FA33, 03FD23 andere Prioritäten (von der Console aus) ein. NB: 03FD23 – 03FA33 = 02F0, das ist die Programmlänge. So können Sie unter Beachtung der Jobtafel auch die Ansprechadressen für Hahn und Hund finden.

L.36.1: Ein durch Exceptionvektor aufgerufener Programmabschnitt wird mit SR = 2000 durchlaufen. Da sich der Prozessor im Supervisor-Modus befindet und in den Status quo ante zurückkehren soll, ist RTE der richtige Befehl.

L.37.1:
a) Beide 03FD76; die einzige zu durchlaufende Subroutine ist CREAJOB.
b) Für F–1 gilt: A7 = 03FCD0, das ist die erste Adresse «unterhalb» (im Bild des Tellerstapels) des Stapels. Der Stapelmechanismus ist also bereit zur Aufnahme eines neuen Eintrags (das wäre die Neuauflage des alten). Der Ablauf wurde also außerhalb von CREAJOB unterbrochen. Weiteren Aufschluß bietet die Aufzeichnung in 03FCB2: Der in Wahrheit nie wieder aktivierte Programmzählerstand 03FD8A wurde beim TRAP #1 abgelegt, der den Job desaktivierte. – Das Entsprechende gilt auch für den F-2-Job.

L.37.2: Mit LEA 22(A0),A6 darf ordnungsgemäße Funktion erwartet werden, auch noch mit 20(A0), denn der Datenbereich bietet mehr Platz als nötig. LEA 26(A0),A6 kann jedoch zum Absturz führen, weil die Längenangabe im Anfang des Job-Kontrollblocks verfälscht wird.

L.37.3: Der künftige Schleifenzähler D4 wurde in 03FDC4 mit Anfangsstand 8 vorbereitet.

L.37.4: Zu 1.: Der Scheduler wird durch Interrupts initialisiert. Das Auftreten von Interrupts (auch höherer Prioritäten) hängt nicht nur von einem Timer, sondern auch von allerlei Zufälligkeiten ab. Zu 2.: Zunächst ein simpler Überschlag: 486 Buchstaben wurden ausgegeben. 486 : 9 = 54, genau (vgl. Abschnitt 37.1) die Anzahl der Tochterprogramme! Jeder F-Job hat also doch wohl genau seine Aufgabe erfüllt! Wo aber wurde behauptet,

L.35.1: Für das Programm Katz gilt: Das unterste Langwort im Stapel ist (auch bei Esel) 00000000. Wegen (3FAD0) = 60E6 soll hier nach 0048' zurückgesprungen werden; es handelt sich also um den Eintrag bei BSR.S AUSGAB. Entsprechend ist 3FADE die beim BSR.S WARTEN abgelegte Rückkehradresse.

Nicht so einfach ist der Eintrag 03FB11 zu deuten. Man könnte vermuten, daß dies die Adresse ist, bis zu der (ausgeschlossen) der Stapel wachsen darf. Dagegen spricht, daß hier noch wesentliche Programmdaten stehen. Wir haben allerdings dem Betriebssystem nicht mitgeteilt, wie weit der Datenbereich gilt. Auch dafür gibt es Möglichkeiten.

L.35.2: Auf das Programm Katz bezogene Deutung:
D0 enthält nach den TRAP-Aufrufen evtl. Fehlermeldungen; hier lief offenbar alles planmäßig.
D1 enthält nach IO_SSTRG die Anzahl der tatsächlich ausgegebenen Zeichen.
D2 enthält die Anzahl der von IO_SSTRG auszugebenden Zeichen («preserved»).
D3 enthält das timeout («preserved»).
D4 dient lediglich der Identifikationsgegenprobe.
D5 enthält nach DBRA (Rotieren des Pufferinhalts) −1.
D6 enthält die Anzahl der um eins nach links verschobenen Bytes.
D7: Offenbar wurde der hohen Wahrscheinlichkeit entsprechend Katz während der Warteschleife suspendiert.
A0 enthält den channel-ID. 002D0004 und 002B0003 sowie die früher benutzten 00000000 und 00010001 lassen eine gewisse Ähnlichkeit erkennen.
A1 wurde zu Beginn der Stringausgabe auf den Pufferanfang gerichtet und dann Byte für Byte inkrementiert. 3FB0E weist auf das erste Byte nach dem Puffer.
A2 vergleichen Sie bitte mit Bild 32.2.
A3 ist permanent auf den Pufferanfang gerichtet.
A4 siehe D4.
A5 wurde vom Programm nicht verändert.
A6 muß beim Verlassen des Programms unverändert an QDOS zurückgegeben werden. Es enthält die Basisadresse, unter der das Programm erreicht wird.

D) Glossar

Absturz. Der Computer ist außer Kontrolle geraten und reagiert nicht mehr (sinnvoll) auf Tastendruck. (Er kann z.B. in eine Schleife geraten sein, in der er das, was über die Tastatur eingegeben werden kann, nicht «versteht»; das Programm kann sinnentstellend zerstört oder Systemvariable mit unsinnigen Werten belegt worden sein.)

Adreßbus. Gruppe von i.a. 16 oder mehr Leiterbahnen, über die bestimmte Speicherstellen, Geräte usw. an-/ausgewählt («adressiert») werden.

analog. Im Gegensatz zu «digital» kommt es auf die Stärke eines Stroms oder die Höhe einer Spannung an (Jedesto-Prinzip). Mikrocomputer arbeiten dagegen digital.

ASCII. American Standard Code for Information Interchange. Codierungsvorschrift, nach der z.B. der Buchstabe «A» intern als $41 codiert wird, die Ziffer «0» als $30 usw.

Basisregister. Enthält eine Basisadresse, von der aus relativ adressiert wird. Realisiert beim M68000 mit der Adressierungsart «Adreßregister indirekt mit Index». Beispiele u.a. in P.14.3, A.5.3 und Kapitel 35: A6 (s. Job-Kontrollblock) zeigt auf den Programmanfang.

Betriebssystem. Ein Programm, das ein Computer zu seiner Selbstverwaltung (Tastaturabfrage, Bildschirmsteuerung, Startadresse usw.) benötigt. Ein Betriebssystem kann sehr einfach sein (Monitorprogramm), aber auch sehr aufwendig.

Binärzahlen. Setzen sich aus den Binärziffern 0 und 1 zusammen. Die Stellenwerte sind Einer, Zweier, Vierer, Achter usw. 1101 1001 = 128 + 64 + 16 + 8 + 1 = 217 = $D9.

Bit. (engl.: Bißchen, auch binary digit); eine Binärstelle im Bitmuster, kann nur einen der beiden Werte 0 oder 1 (L oder H) darstellen. Einer Leiterbahn eines Busses entspricht z.B. auch ein Bit.

Bitmuster. Man braucht die Nullen und Einsen eines Datums nicht unbedingt als Zahl aufzufassen; sie können auch eine andere Bedeutung haben. Man spricht dann ganz neutral von Bitmustern.

Breakpoint. Eine Adresse, bei der (z.B. zur Fehlersuche) der Programmablauf (vorübergehend) unterbrochen werden kann.

Buffer. Bei uns Zwischenspeicher im weiteren Sinn. Vgl. Puffer.

Byte. Eine Gruppe von 8 bit; kann Zahlen von 0 bis 255 (bzw. $00 bis $FF) repräsentieren.

Cartridge. Mikro-Bandkassette für den QL.

channel-ID. Siehe Kanalkennzahl.

Cleancomputer. Das residente Betriebssystem ist äußerst knapp und enthält u.U. nur ein Urladeprogramm, das das eigentliche Betriebssystem lädt, oder ein etwas komfortableres, aber dennoch ziemlich spartanisches Monitorprogramm. Cleancomputer sind übersichtlich und flexibel. Man kann verschiedene Sprachen laden, manche sogar in Compiler- und/oder Interpreterversion.

Compiler. Übersetzt den in einer höheren Sprache geschriebenen (problemorientierten) Quelltext vor Programmablauf ein für allemal in ein Maschinenprogramm. Vgl. Interpreter.

CPU. Central Processing Unit, der Mikroprozessor.

daß ein und derselbe F-Job immer ein und denselben Kennbuchstaben ausgibt? F–1 läuft vielmehr so ab:
Ausgabe 'A',
  CREAJOB mit Inkrementierung von D5;
Ausgabe 'B',
  CREAJOB mit Inkrementierung von D5;
Ausgabe 'C' usw. ...
Die Tochterprogramme geben also so aus:

```
F–1:  ABCDEFGHI
F–2:   BCDEFGHIJ
F–6:    CDEFGHIJK
F–3:    CDEFGHIJK
F–4:     DEFGHIJKL
F–5:      EFGHIJKLM
F–7:       FGHIJKLMN
```

Die vollständige Rekonstruktion darf interessierten Lesern überlassen werden.
Zu 3.: Die Antwort geht aus «Zu 2.» hervor.

L.39.1: MLDUFN läßt sich entnehmen, daß die Mühlen $20 bzw. $35 bzw. $49 durch die Felder 00, 04, 08, 0C bzw. 07, 16, 25, 34 bzw. 03, 16, 29, 3C verlaufen; im Klartext: $20 verläuft durch die senkrechte vordere linke Kante, $35 verläuft in der zweiten Schicht von oben von der vorderen rechten zur hinteren linken Kante, und $49 ist die Raumdiagonale durch die Ecke oben rechts vorn.

L.39.2: Wenn das entsprechende Feld unbesetzt ist, wird der aus dem Spielwert im MSN und «1» im LSN zusammengesetzte Eintrag aus SPIELF nach D4 geholt und mit D2 verglichen. Ist der Inhalt von D4 größer als der von D2, übernimmt D2 den höheren Wert, enthält also am Schluß das Maximum.

L.39.3: Angenommen, der Computer hat als nächsten Zug die Besetzung von Feld $27 ermittelt. Dann enthält D0 in binärer Notierung 0010 0111, was sich unmittelbar in Ebene, Zeile, Spalte zerlegen läßt: xx 10 01 11. Wie Bild 39.2 zeigt, ist für Feld $27 E = 3, Z = 2 und S = 4. Man erhält diese Zahlen, indem man die Dupel um 1 erhöht und zum ASCII aufbereitet, was in AUFBZC (AUFBereiten Zug Computer) durch Addition von $31 in einem Gang erledigt wird. Das ROL.B #2,D0 dürfte nun verständlich sein.

zugeordnet, unter denen sie künftig (wie der Fernsprechteilnehmer unter seiner Rufnummer) erreicht werden können. Der Kanal übernimmt die Feinarbeit der Geräteansteuerung (vgl. Driver).

Label. Eine Variable, die nur zwei Zustände annehmen kann. Anfangs ist sie leer. Nach der Wertzuweisung kann der Inhalt i.a. nicht mehr verändert werden (beim NDR-Computer jedoch mit einem weiteren EQU), kann aber wie bei einer normalen Variablen immer wieder abgefragt und in Rechnungen einbezogen werden.

Linefeed. Zeilenvorschub; normalerweise codiert $0A.

Linken. Ein neues Glied in eine Kette einfügen; hier: ein neues Programm oder einen zusätzlichen Speicherbereich in den Gesamtablauf einbeziehen.

lokal ist eine Variable, die nur in einem bestimmten Programmteil (z.B. einer Prozedur) existiert.

LSB. Lower Significant Byte, niederwertigstes Byte (z.B. in einem Wort oder Langwort).

LSb. Lower Significant Bit.

LSN. Lower Significant Nibble.

Mikrodrive. Miniatur-Bandlaufwerk des QL.

Mnemonik. Eine gut zu behaltende Kurzformulierung für ein farbloses Abstraktum; z.B. «RTS» (ReTurn from Subroutine) statt «4E75».

Monitor. (lat.: Überwacher). a) Datensichtgerät. b) einfaches Betriebssystem (s.d.).

MSB. Most significant byte, höchstwertiges Byte (z.B. in Wort oder Langwort).

MSb. Most significant bit.

MSN. Most significant nibble.

Negative Zahl. Eine ganze Zahl gilt häufig als negativ, wenn ihr höchstwertiges Bit gesetzt ist: $-128 = \$80 < \$7F = +127$.

Nibble. Halbbyte, eine Gruppe von vier Bits.

Offset. (Sehr frei übersetzt: Aufgesetztes). Eine Adreßdistanz, die (i.a. vorzeichengerecht) zu einer Basisadresse zu addieren ist.

Opcode. Kurzform für «Operations-Code»; z.B. 4E75 für RTS.

Pixel. Bildpunkt, kleinstes Element der Bildausgabe.

Plotter. Zeichengerät, dessen Schreibstift vom Computer in X- und Y-Richtung bewegt und auch gehoben oder gesenkt werden kann.

Pointer. (engl. Zeiger). Meistens ein Register, das eine Adresse enthält, also auf eine bestimmte Speicherstelle zeigt.

Puffer. Im engeren Sinne ein Zwischenspeicher, in den ein schnelleres Gerät (z.B. der Computer) Daten ablegt, die ein langsameres Gerät (z.B. der Drucker) allmählich abarbeitet. Dadurch wird das schnellere Gerät für andere Aufgaben frei.

RAM. Random Access Memory, Schreib-/Lesespeicher, z.B. zur Unterbringung von Variablen und nicht-residenten Programmen.

resident ist ein im ROM stehendes Programm, denn es ist immer vorhanden und braucht nicht erst geladen zu werden. Viele Betriebssysteme sind resident.

ROM. Read Only Memory, Festwertspeicher; der Anwender kann nur aus ihm herauslesen.

Scheduler. Der Teil eines Mehrbenutzer-Betriebssystems, der (im Zusammenwirken mit der Hardware) den einzelnen Jobs Rechenzeit zuweist (s. Abschnitt 31.1 und Kapitel 35).

sedezimal. Anderes Wort für hexadezimal.

CR, cr. Carriage Return, Wagenrücklauf bei Schreibmaschinen (bzw. Fernschreibern). Mit der CR-Taste (Code $0D) werden Eingaben abgeschlossen. Vgl. Enter.

Datenbus. Eine Gruppe von i.a. 8 oder 16 Leiterbahnen, über die digital codierte Daten transportiert werden.

Datum. Einzahl von Daten.

digital. Im weiteren Sinne «durch Ziffern dargestellt»; im engeren Sinne «durch Binärzahlen darstellbar», d.h. durch eine Gruppe von Leitern, die entweder «Impuls» (1) oder «keinen Impuls» (0) führen.

Digitalrechner. Arbeitet mit Daten, die durch die 0/1-Zustände mehrerer elektronischer Bauelemente (Drähte, Leiterbahnen, Transistoren, Kondensatoren usw.) codiert sind.

DOS. Disk Operating System, Diskettenbetriebssystem.

Driver. Hier Routinen, die nach der Übergabe der wesentlichen Parameter die hardwarespezifische Feinarbeit zur Ansteuerung der Ein-/Ausgabegeräte übernehmen.

Echo. Über den Bildschirm wird ausgegeben, über die Tastatur zunächst einmal nur eingegeben. Wenn aber die Eingabe auch auf dem Bildschirm protokolliert wird, nennt man das Echo.

ENTER. (engl., u.a. aufnehmen); andere Bezeichnung/Beschriftung der CR-Taste (s.d.).

Flag. (Signal-)Flagge, ein Bit i.a. im Innern des Prozessors, das durch seinen Inhalt anzeigt, ob ein bestimmtes Ereignis (wie etwa Vorzeichenwechsel oder Überlauf des Rechenwerks) eingetreten ist.

FORTH. Eine prozessornahe und deshalb sehr schnelle Programmiersprache mit schon recht angenehmem Komfort. Gerechnet wird im Zahlenstapel. Die Befehlsstruktur entspricht der UPN (s.d.).

FP. Abkürzung für Floating Point, Fließkomma(-zahl). Siehe Kapitel 22 bis 27.

global. Ist eine Variable, die im ganzen Programm gilt; bei uns auch auf Pointer angewendet. Gegenteil: lokal.

Hexadezimalzahlen haben die Stellenwertbasis 16: Einer, 16er, 256er, 4096er usw. Sie werden nach Motorola-Norm durch vorangestelltes «$» gekennzeichnet. $2468 = 2*4096 + 4*256 + 6*16 + 8*1 = 9320.

Hexadezimalziffern sind zusätzlich zu den Dezimalziffern die Zeichen A=10, B=11, C=12, D=13, E=14 und F=15. $FD = 15*16 + 15*13 = 253. Siehe auch negative Zahlen.

Hexdump. Speicherauszug (dump) in hexadezimaler Notierung.

ID. Identifier, Kennzahl.

Index. Beim M68000 ein Daten- oder Adreßregister, dessen Inhalt vorzeichengerecht zu einer in einem Adreßregister stehenden Basisadresse addiert wird.

Interpreter übersetzen einen in einer höheren Programmiersprache geschriebenen (problemorientierten) Quelltext während des Programmablaufs. In Schleifen wird jede Anweisung immer wieder aufs neue übersetzt; das kostet Zeit.

Job (Arbeit). Hier: unabhängig voneinander ablaufende Programme.

Job-Kontrollblock. Speicherbereich, in den z.B. bei Unterbrechung durch den Scheduler die wesentlichen Angaben über das suspendierte Programm (ab wo muß es mit welchen Registerinhalten aktiviert werden?) «gerettet» werden (s. z.B. 35.2).

Kanalkennzahl. Den verschiedenen Bildschirmfenstern, der Tastatur, den Mikrodrives usw. werden Zahlen (channel-ID)

E) Modus-Experiment mit dem NDR-Computer

Der NDR-Computer befindet sich normalerweise im Supervisor-Modus. Man kann ihn in den User-Modus mit korrekter Rückkehr ins Grundprogramm zwingen, wenn man den in Kapitel 33 dargestellten Weg konsequent umkehrt:

Nachdem der Inhalt des SSP nach D1 und (unnötigerweise) die dort eingetragene Rücksprungadresse nach D2 gerettet wurden, werden ein neuer Systemstapel ab SSTAPL und ein Userstapel ab USTAPL eingerichtet; beide wachsen zu niedrigeren Adressen hin. MARKIR wird keiner Erläuterung mehr bedürfen.

Vor dem Übergang in den User-Modus wird die Adresse, in der der User-Programmtorso beginnt, mit PEA auf dem momentan gültigen Stapel (das ist z. Zt. noch SSTAPL) abgelegt, danach der Inhalt des Statusregisters, aus dem sodann das S-Bit wegmaskiert wird. Nun versetzt RTE den Prozessor in den User-Modus.

Die Rückkehr in den Supervisor-Modus bewirkt ein TRAP #8 mit Vektor $008012 (im ROM). In $008012 (RAM) steht aber JMP 003230. Der erste Befehl unseres Programms verbiegt diesen Soft-Vektor, der nun zum Programmabschnitt SUPER2 führt, der ebenfalls seine Spur im Puffer hinterläßt, den ursprünglichen Systemstapel wiederherstellt und wie immer ins Grundprogramm zurückführt.

Nach Durchlauf sieht der Arbeitsbereich so aus:

```
                    >>>>
BUFFER:  02004E  2 7 0 0  00020088
         020054           00020022
         020058  0 7 0 0  00020074
         02005E           00020040
         020062  ##
         020064  2 7 0 0  00020082
         02006A           00020036
         02006E  8 8 8 8  88888888
         020074           00020040
USTAPL:  020078  <<>>
         02007C  8 8 8 8  88888888
         020082           00020036
         020086  0 7 0 0  00020042
SSTAPL:  02008C  <<<<     22222222
```

signed numbers. Vorzeichenbehaftete Zahlen, z.B. $80 = −128. Gegenteil: unsigned numbers. Siehe auch negative Zahlen.

Speicher. Vorrichtung, in der als elektrische (oder magnetische) Impulse codierte Daten aufbewahrt bzw. bereitgestellt werden können.

Stack (Stapel). Ein Speicherbereich, in dem vor allem Rücksprungadressen, aber auch andere Daten vorübergehend abgelegt werden. Siehe auch Stackpointer.

Stackpointer, Stapelzeiger. Ein Hinweisregister, das auf die momentan gültige Adresse im Stapel zeigt; das ist i.a. die, in der der zuletzt eingeschriebene oder als erster herauszulesende Eintrag steht. Beim M68000 ist A7 der Stackpointer.

Stapel. Siehe Stack.

Stapelzeiger. Siehe Stackpointer.

String. Zeichenkette. Wird z.B. die Hexadezimalzahl 1A2E zum String aufbereitet, sieht das intern so aus: 31 41 32 45.

Systemabsturz. Siehe Absturz.

Systema-DOS. Ein Diskettenbetriebssystem der Firma Systema-Soft.

Systemvariable. Variable, die Angaben über das Computersystem enthalten: Wie viele User sind angeschlossen? Wo steht der Cursor? Ausgabe auf Drucker oder Bildschirm? Letzte gedrückte Taste? usw.

Taktzyklen. Digitalrechner arbeiten nach einem vorgegebenen Takt, der von einem Schwingquarz erzeugt wird. Wird der Prozessor mit 8 MHz getaktet, so erzeugt der Quarz 8000000 (oder sogar 16000000) Taktzyklen (Schwingungen) je Sekunde.

Task. Hier eine von einem Computer zu übernehmende Aufgabe, Abarbeitung eines Programms bzw. das Programm selbst.

TOS. Top of stack, das sozusagen aktive Ende des Stapels. Hier ist der letzte Eintrag abgelegt bzw. wird von hier die Rücksprungadresse geholt.

transient. Vorübergehend. Transient program area: Bereich des RAM, in dem die vorübergehend geladenen Programme untergebracht werden können.

unsigned numbers. Vorzeichenlose Zahlen; das MSb stellt einen Zahlenwert und nicht etwa das Vorzeichen dar: $FFFF = +65536.

UPN. Umgekehrte Polnische Notation. Nach Jan LUKASIEWICZ so benannte Schreibweise, die 7 − 4 als 7 4 − notiert.

vorzeichenbehaftete Zahlen. Siehe signed (und unsigned) numbers.

Stichwortverzeichnis

@AUFXY 208
@CI 62
@CLRSCREEN 64
@CO2 62
@CRLF 64
@CRT 134
@CSTS 102
@DELAY 64
@ERAPEN 213
@FIRSTTIME 208
@LST 134
@PRINT8D 182
@PRINTnX 77
@READ 112
@SCHREITE 208
@SETFLIP 64
@SETPEN 213
@WERT 90, 144
@WRITE 64

A
ABCD 97, 153
Absolut-kurz-Adressierung 27, 62
Absolut-lang-Adressierung 26
ADD, ADDA, ADDI 77, 153, 154
Additionsbefehle 153
ADDQ 60, 77, 154
ADDX 154
Adreßbus 20, 332
Adreßdistanz 29
Adresse, effektive 23
Adressierung programmzähler-relativ 30
Adressierung speicherplatzunabhängig 29, 33
Adressierungsarten 19, 23
Adreßregister indirekt mit Index 40
– – mit Postinkrement 33
– – mit Predekrement 34
aktiver Job (QL) 283
AND 86
ANDI to SR 272
Anmeldung eines Programms beim Betriebssystem QDOS 275
Arbeitsbereich des Betriebssystems 206
ASCII-Code/-String 75
ASL. ASR 84
Assembler 39
– bedienen, K-Seka (Amiga) 167
– bedienen, Metacomco (QL) 71
– bedienen, RDK (NDR-Computer) 55
Assemblerdirektive 42
Assemblerprotokoll 57
Assemblieren 39
Aufzeichnen auf Diskette 133
Ausgabe
– von Zeichen (NDR) 62
– von Strings (NDR) 64
– von Strings (QL) 66
Ausnahmen vom normalen Programmablauf 259, 265
Ausnahmezustand 181
Autovektor-Interrupt 261

B
BASIC-Hostprogramm 17, 18, 206
Basisregister 82
Bcc 53
BCHG 254
BCLR 102, 254

```
020000  23FC 0002002E    VORBER: MOVE.L   #SUPER2,$080014
020006  00080014
02000A  220F                     MOVE.L   A7,D1
02000C  2417                     MOVE.L   (A7),D2
02000E  2E7C 0002008C            MOVEA.L  #SSTAPL,A7
020014  41FA 0062        SUPER1: LEA      USTAPL(PC),A0
020018  4E60                     MOVE     A0,USP
02001A  41FA 0032                LEA      BUFFER(PC),A0
02001E  6100 0022                BSR      MARKIR
020022  487A 0018                PEA      USERMD(PC)
020026  40E7                     MOVE     SR,-(A7)
020028  0257 DFFF                ANDI.W   #$DFFF,(A7)
02002C  4E73                     RTE
02002E
02002E  30FC 2323        SUPER2: MOVE.W   #'##',(A0)+
020032  6100 000E                BSR      MARKIR
020036  2E41                     MOVEA.L  D1,A7
020038  2E82                     MOVE.L   D2,(A7)
02003A  4E75                     RTS
02003C
02003C  6100 0004        USERMD: BSR      MARKIR
020040  4E48                     TRAP     #8
020042
020042  40D8             MARKIR: MOVE     SR,(A0)+
020044  20CF                     MOVE.L   A7,(A0)+
020046  20D7                     MOVE.L   (A7),(A0)+
020048  4E75                     RTS
02004A
02004A  3E3E3E3E                 DC.B     '>>>>'
02004E                   BUFFER: DS.B     42
020078  3C3C3E3E         USTAPL: DC.B     '<<>>'
02007C                           DS.B     16
02008C  3C3C3C3C         SSTAPL: DC.B     '<<<<'
020090
020100                           ORG      $20100
020100  41FA FF4C        BEQUEM: LEA      BUFFER(PC),A0
020104  703D                     MOVEQ    #$3D,D0
020106  10FC 0088        LOOP:   MOVE.B   #$88,(A0)+
02010A  51C8 FFFA                DBRA     D0,LOOP
02010E  23FC 3C3C3E3E            MOVE.L   #'<<>>',USTAPL
020114  00020078
020118  41E8 0004                LEA      04(A0),A0
02011C  7073                     MOVEQ    #$73,D0
02011E  10FC 0022        LOOP2:  MOVE.B   #$22,(A0)+
020122  51C8 FFFA                DBRA     D0,LOOP2
020126  4E75                     RTS
020128                           END
```

H
Hanoi, Türme von 189
Hexdump 15
Hornersches Schema 90, 93

I
I/O-Ports 27
Inaktiver Job (QL) 283
Indexregister 40
Interrupt 259
Interrupt-Autovektor 261
Interrupt-Ebenen 260, 263
– -Maske 260
IO_CLOSE 291
IO_FBYTE 104
IO_FLINE 288

J
JMP 52
Job als selbständiges Programm (QL) 295
Job-Kontrollblock 113, 206, 280, 281, 293
Jobtafel (QL) 293
JSR 58

K
K-Seka-Assembler 167
Kanalkennzahl 66, 104, 277
Kommentar (Assembler) 42
Komplementarität Mensch–Computer 306
Konstantenadressierung 26
Kontrollbus 332
Konvertierung s. a. Umwandlung
– DEZ → FP 142
– FP → DEZ 146
Konvertierungsfehler 150

L
Label 42
LBYTES (QL) 206
LEA 29
LINK 256
LSL, LSR 84

M
Mantisse 136
Maskieren 86
Mehrbenutzersystem (s.a. Multitasking) 19, 265
Mikroprozessor 331
Mnemonik 37, 214
Monitorprogramm 15
MOVE 26, 38
MOVE to CCR 186
MOVE to SR 48, 272
MOVE to USP 271
MOVEA 42, 125
MOVEM 31, 88
MOVEP 258
MOVEQ 30, 39
MT_CJOB 297
MT_FRJOB 292
MT_PRIOR 277
MT_TRAPV 286
MULS 175
Multiplikation mehrstellig 176
Multiplikationsbefehle 175
Multitasking 265, 275, 279
MULU 92, 175

N
N-Flag 48
NBCD 87
NEG 87
negative Zahl 27
NOP 59
NOT 87

O
Objektcode 57
Offset 42
Opcode 37
OR 86
ORG-Direktive 45, 56, 71
ORI to SR 272

P
PC (Program Counter) 29
PEA 254
Postinkrement 33

Bedingungsschlüssel und Flags 159
Betriebssystem, Arbeitsbereich 206
Binärbrüche 140
BLK-Direktive 170
BRA 52
BSET 254
BSR 58
BTST 145
Bubblesort 104

C
C-Flag (carry) 47
CALL (Amiga) 17, 31
CALL (QL) 18, 206
Carry 47
CCR 46, 47, 155
Channel-ID (s. a. Kanalkennzahl) 68
CLI 167
CMP 48
CMPI 46
CN_ITOHB, ITOHL, ITOHW 80
Codierung Bcc 53
Codierung BRA 52
– DBcc 51
– MOVE 38
– MOVEQ 39
Command Line Interface (Amiga) 167
Condition Code Register 46, 47
Console 288
Console-Fenster 104

D
Datenbus 19, 332
Datenformate 20
DBcc 45, 51
DBEQ 46
DBF 44
DBRA-Schleife 43
Default-Wert 55
Disassembler 39, 214
Diskette, Aufzeichnung (Systema-DOS) 133
Division, bitweise 184
Divisionsbefehle 181
DIVS 181
DIVU 148, 181
DROP 142
DS-Direktive 42, 123
DUP 142

E
Editor (Amiga) 168
– (NDR) 55
– (QL) 72
Effektive Adresse 23
Eingabe (NDR) 62, 112
– (QL) 104, 288
Einzelschrittmodus (s. a. Tracing) 57, 284
EOR 87
EORI to SR 272
EPROM 331
EQU-Direktive 71
Eratosthenes, Sieb des 99
EVEN-Direktive 123
Exception 181, 265
EXEC (QL) 72, 206
EXG 78
Exponent 136
EXT 117
extend bit (X-Flag) 48

F
Flagkombinationen 49
Flags bei Addition 155
– bei Subtraktion 158
Flags des CCR 47, 155
– des SR 259
Flags und Bedingungsschlüssel 159
Fließkomma-Addition 151
– -Division 181
– -Multiplikation 175, 178
– -Subtraktion 162
Fließkommazahlen 136
Floating Point 136
FORTH 141
FP siehe Floating Point 136

G
Grafische Darstellung 193

Tochterprogramme 295
TOS (Top of stack) 146, 163
Trace-Modus, Tracing 284
TRAP 104, 133, 265, 266, 268, 270
TRAP-Vektoren umlenkbar (QL) 282
TST 106
Türme von Hanoi 189

U
Überlauf 47
Übertrag 47
Umwandlung
– von einem Zahlensystem ins andere 90
– DEZ → HEX 90
– HEX → DEZ 93
– HEX → String 75
UNLK 256
unsigned numbers 157
UPN 142, 163
Usermodus 265
– (NDR) 274
User-Stackpointer, USP 265

UT_MINT 104
UT_MTEXT 68
UT_SCR 66, 268

V
V-Flag 47
vektorisierte Routinen 68, 268
vorzeichenlose Zahlen 157

W
Wurzelziehen 288

X
X-Flag (extend bit) 48

Z
Z-Flag 47
zählerkontrollierte Schleifen 43
Zahlen mit/ohne Vorzeichen 27, 157
Ziel 37, 42
Zweierkomplementarithmetik 50
Zwei-Pass-Assembler 42, 73

Predekrement 25, 34, 334
Primzahlen 99, 182
Priorität 277, 279, 282
Privilegierte Befehle 254, 271
Programm (s. a. Job) 295
Programmierung eines Mikroprozessors 331
Programmschleife 43
Programmzähler 29
programmzählerrelativ (Adressierung) 30, 72

Q
QDOS-Version 68, 69
Quelle 37, 42
Quelltext 55

R
RAM 331
Raummühle 304
Register 23, 24
Registerliste 88, 242
Rekursion, rekursiv 189
relokatibel 29, 72
Relozieren 32
Reset (NDR) 57
RESET 271
RESPR (QL) 16
ROL, ROR 82, 84
ROM 331
Rotieren und Shiften 83
ROXL, ROXR 84
RTE 262, 270, 271
RTS 58
Rücksprung
– nach Amiga-DOS 172
– nach QDOS 28

S
S-Bit 265
S-Mode 288
SADD-Funktion (Amiga) 16
Scheduler 277, 279, 282
Schleifen mit Zweitbedingung 45
Screen driver (QL) 104
Screen-Fenster 104
SD_CLEAR 195
SD_FILL 194
SD_SCROL 104
SD_TAB 104
Selbstentfernung eines Programms (QL) 291
Self-cloning Program 293, 303
Shiften 83
Signed numbers 157
Single step mode (s. a. Einzelschritt) 284
Sortieren 110
Speicher 331
speicherplatzunabhängige Adressierung 29, 33
Sprünge, bedingt und unbedingt 52
SSP 265
Stack (s. a. Stapel) 58
Stackpointer (Stapelzeiger) 59
Stapel und Subroutinen 58
Status eines Jobs/Programms 206, 283, 295
Statusregister 259
Steuerbus 332
STOP 271
Strategie 306
Stringausgabe (QL) 66
SUB, SUBA, SUBI, SUBQ, SUBX 80, 162
Subroutinen 58
Subtraktion und Flags 158
Subtraktionsbefehle 162
Supervisor-Bit 260
Supervisormodus 265, 274
Supervisor-Stackpointer 265
suspendierter Job (QL) 283
SV_JBBAS 280
SWAP 90
symbolische Adressen 42
Symboltabelle 243, 250
Systemvariable 113, 280

T
T-Bit 260, 286
Tabelle absuchen 95
Taktgenerator 331

VOGEL
Computerbücher

«CHIP-WISSEN kurz und bündig»

Biethan:
MS-DOS/PC-DOS
Das Betriebssystem für IBM PC und Kompatible
ISBN 3-8023-0866-2

Diemer: **CAD**
Zeichnen und Konstruieren mit dem PC
ISBN 3-8023-0892-1

Diemer:
Lokale Netzwerke
Von der Insellösung zum Rechnerverbund
ISBN 3-8023-0789-5

Diemer: **Relationale Datenbanken**
Sicherheit, Integrität und Unabhängigkeit in der Datenverarbeitung
ISBN 3-8023-0797-6

Förster: **Word**
Textverarbeitung mit dem IBM PC und Kompatiblen
ISBN 3-8023-0893-X

Langfelder: **WordStar**
Textverarbeitung mit dem Mikrocomputer
ISBN 3-8023-0798-4

Mahnke:
Software Engineering
Planung, Methoden, Darstellungstechniken
ISBN 3-8023-0886-7

Overbeck: **Kalkulationsprogramme**
Grundlagen, Anwendungen, vergleichende Übersicht
ISBN 3-8023-0826-3

Schwinn: **dBase II**
Einstieg in das Datenbanksystem für Mikrocomputer
ISBN 3-8023-0800-X

Schwinn: **dBase III**
Einstieg in das Datenbanksystem für den IBM PC und Kompatible
ISBN 3-8023-0853-0

Teege:
Integrierte Software
1-2-3, Symphony, Knowledge Man, Open Access und Framework im Vergleich
ISBN 3-8023-0639-2

Diese Bücher erhalten Sie beim Buchhandel oder direkt beim Vogel-Buchverlag Würzburg, Postfach 67 40, 8700 Würzburg 1.

Sie erhalten bei Ihrem Buch- und Computerfachhändler kostenlos das neue Verzeichnis „Vogel-Computerbücher"

VOGEL-BUCHVERLAG
WÜRZBURG

VOGEL-Computerbücher — mehr wissen, mehr leisten.

PC: Damit Sie einfach mehr von Ihrem Personal-Computer haben!

PC Personal Computer + PC Soft Anwender-Programme, gibt Ihnen alle Informationen, die Sie brauchen, um Ihren PC bestmöglich zu nutzen.

PC bringt für Sie: Kaufberatung, Marktübersichten, Tests, Beratung bei der Einsatzplanung, Anwendungsbeispiele, Problemlösungen, Erfahrungsberichte aus der Praxis.

PC gibt es für 7,— DM monatlich bei Ihrem Zeitschriftenhändler. Im Abonnement für 5,50 DM pro Heft. Oder direkt beim PC-Leser-Service, Vogel-Verlag, Postfach 67 40, 8700 Würzburg 1.

VOGEL Computerbücher

Langfelder, Christian

WordStar

kurz und bündig

Reihe CHIP WISSEN

100 Seiten, 22 Abb.

ISBN 3-8023-**0798**-4

Diese leichtfaßliche Einführung in WordStar ist für alle unentbehrlich, die mit diesem Textverarbeitungsprogramm arbeiten wollen, sich aber nicht unnötig lange mit dem Drumherum belasten können. Die wichtigsten WordStar-Befehle werden sehr anschaulich und weitgehend rechnerunabhängig dargestellt.

Teege, Frank
Hoffmann, Michael

Schneller erfolgreich

mit Lotus Symphony

Integrierte Software als Universalwerkzeug

Reihe CHIP WISSEN

228 Seiten, 78 Abb .

ISBN 3-8023-**0865**-4

Dieses Buch stellt zunächst die benötigte Hardware vor und erklärt dann die Installation von Symphony sowie das Tutor-Programm und die Hilfe-Funktionen. Anschließend werden die fünf Hauptteile des Programms beschrieben. Durch viele Beispiele und Erklärungen aller zur Verfügung stehenden Befehle lernt der Leser, selbständig auch schwierige und umfangreiche Arbeiten mit Symphony zu lösen. Die Erweiterungen der neuesten Version 1.1 werden soweit wie möglich bereits berücksichtigt.

Teege, Frank

Integrierte Software

kurz und bündig

1-2-3, Symphony, Knowledge Man, Open Access und Framework im Vergleich

Reihe CHIP WISSEN

156 Seiten, 86 Abb.

ISBN 3-8023-**0639**-2

Dieses Buch stellt die in letzter Zeit so populär gewordenen integrierten Software- oder Multifunktionspakete vor. Es wendet sich an Anwender, die vor dem Kauf anhand einer vergleichenden Übersicht entscheiden wollen, welches Programm für ihre Belange am geeignetsten ist.

Biethan, Gunter

MS-DOS/PC-DOS

kurz und bündig

Das Betriebssystem für IBM PC und Kompatible

Reihe CHIP WISSEN

144 Seiten, 10 Abb.

ISBN 3-8023-**0866**-2

Dieses Buch baut dem Anwender eine Brücke zum besseren Verständnis und zur leichteren Bedienung seines PCs, um ihm so den Weg zur aktiven, erfolgreichen Computerei zu ebnen. Es soll nicht das Handbuch ersetzen, sondern geht gezielt auf typische Anwenderprobleme und -fragen ein, wie z.B. Kompatibilitätsprobleme, Unterschiede zwischen den verschiedenen Versionen von MS-DOS u.a.m. Dem Wunsch vieler Anwender von MS-DOS folgend, wurde die Beschreibung der nicht zum Betriebssystem gehörenden Hilfsprogramme „EDLIN" und „DEBUG" in dieses Buch mit aufgenommen.

Diese Bücher erhalten Sie beim Buchhandel oder direkt beim Vogel-Buchverlag Würzburg, Postfach 67 40, 8700 Würzburg 1.

Sie erhalten bei Ihrem Buch- und Computerfachhändler kostenlos das neue Verzeichnis „Vogel-Computerbücher"

VOGEL-BUCHVERLAG
WÜRZBURG

VOGEL-Computerbücher — mehr wissen, mehr leisten.

VOGEL
Computerbücher

Sacht, Hans-J.

Daten, Disketten, Dateien

Reihe CHIP WISSEN
300 Seiten, 83 Abb.
ISBN 3-8023-**0751**-8

Wer anspruchsvollere Programme in BASIC erstellen will, muß mit Diskettenspeicherung arbeiten. Der Verfasser erklärt, wie Betriebssysteme funktionieren und wie man Dateien aufbaut; er hilft allen, die Programme entwickeln wollen und dabei die Verarbeitung extern gespeicherter Daten benötigen. Hinweise zum Benutzen verschiedener Diskette-Betriebssysteme runden das Buch ab.

Mahnke, Hans

Software Engineering kurz und bündig

Planung, Methoden, Darstellungstechniken
Reihe CHIP WISSEN
112 Seiten, zahlr. Abb.
ISBN 3-8023-**0886**-7

In diesem Buch wird verständlich beschrieben, wie bei der Softwareentwicklung folgende Kriterien erfüllt werden können:
- Verläßlichkeit in bezug auf technische Stabilität als auch auf ordnungsgemäßes Erfüllen der fachlichen Aufgaben
- Benutzerfreundlichkeit und Wiederverwendbarkeit
- Übertragbarkeit auf andere Systeme

Füller, Klaus

Host On Line

Mit dem PC an Datenbank und Mailbox
Reihe CHIP WISSEN
176 Seiten, 26 Abb.
ISBN 3-8023-**0879**-4

Mit dem Arbeitsplatzrechner läßt sich auf ein immer breiteres Informations- und Kommunikationsangebot in Form von Datenbanken und Mailboxen zurückgreifen. Wie können Sie Ihren PC an einen Minicomputer oder Großrechner anschließen? Dieses Buch enthält praktikable Vorschläge zur Realisierung der Rechner-Rechner-Kommunikation und beschreibt die dazu benötigte Software. Eine Sammlung von Beispielen und Adressen ermöglicht erste Kontakte mit der Welt der Abfragesprachen und Computerkonferenzen.

Förster, Hans-Peter
Rompel, Helmut

Der Heimcomputer als Btx-Terminal

Wie man Btx-Informationen mit dem HC speichert und weiterverarbeitet
Reihe HC – Mein Home-Computer
196 Seiten, zahlr. Abb.
ISBN 3-8023-**0850**-6

Dieses Buch informiert über das Btx-System, beschreibt die Anwendungsmöglichkeiten, erklärt, wie man selbst zum Btx-Teilnehmer wird, und gibt Anregungen in Form von Programmbeispielen anhand des Commodore 64, wie die Btx-Nutzung automatisiert und erweitert werden kann.

Diese Bücher erhalten Sie beim Buchhandel oder direkt beim Vogel-Buchverlag Würzburg, Postfach 67 40, 8700 Würzburg 1.

Sie erhalten bei Ihrem Buch- und Computerfachhändler kostenlos das neue Verzeichnis „Vogel-Computerbücher"

VOGEL-BUCHVERLAG
WÜRZBURG

VOGEL-Computerbücher – mehr wissen, mehr leisten.