```
*************************
*                       *
*  RUNTIME - BIBLIOTHEK *
*    für NDR - PASCAL   *
*                       *
*************************

*****************************************
*                                       *
*  Macros zur Erzeugung von Objektdateien *
*                                       *
*****************************************
* Rev. 1.1
* (C) 1989,1990 Klaus Rumrich
*****************************************

macro name
_nam||:
dc.b '|0',0
org _nam||+16
endmacro


* .LEA label,Ax

macro lea
lea |0(pc),|1
adda.l (|1),|1
endmacro


* .MODUL Modulname

macro modul
_beg||:
dc.w 1
dc.w 22
.name |0
dc.l _end||-_beg||
dc.b 0          * CPU
dc.b 0          * Reserve
endmacro


* .ENDMODUL

macro endmodul
dc.w 2,0
_end||:
endmacro


* .CODE         (Anfang Maschinencode)

macro code
dc.w 3,_doc||-pc_stand
_cod||:
offset _cod||-pc_stand
org pc_stand
_ocd||:
endmacro


* .ENDCODE      (Ende Maschinencode)

macro endcode
pc_stand newequ *
_doc||:
org pc_stand+_cod||-_ocd||
offset 0
endmacro
```

```
* .GLOBPROC label

macro globproc
dc.w 4,20
.name |0
dc.l |0
endmacro


* .EXTPROC label

macro extproc
dc.w 5,24
.name |0
dc.l |0
dc.l |0+6
.code
|0:              * Hilfslabel
pea |0(pc)
add.l #0,(a7)
rts
.endcode
endmacro


* .EXTPROC2 label,interner-name

macro extproc2
dc.w 5,24
.name |0
dc.l |1
dc.l |1+6
.code
|1:              * Hilfslabel
pea |1(pc)
add.l #0,(a7)
rts
.endcode
endmacro


* .GLOBLAB label

macro globlab
dc.w 6,20
.name |0
dc.l |0
endmacro


* .EXTLAB label

macro extlab
dc.w 7,16+6
.name |0
dc.b 0,4        * PC-rel, Long
dc.l |0
.code
|0:              * Hilfslabel
dc.l 0
.endcode
endmacro


* .EXTLAB2 label,interner-name

macro extlab2
dc.w 7,16+6
.name |0
dc.b 0,4        * PC-rel, Long
dc.l |1
.code
|1:              * Hilfslabel
```

```
dc.l 0
.endcode
endmacro


* .EXTCONST label

macro extconst
dc.w 7,16+6
.name |0
dc.b 2,4        * absolut, Long
dc.l |0
.code
|0:            * Hilfslabel
dc.l 0
.endcode
endmacro


* .ENDLIB

macro endlib
dc.w 12,0
endmacro


* .GLOBCONST const

macro globconst
dc.w 13,20
.name |0
dc.l |0
endmacro


* .SEGMENTC Segmentname, Groupname

macro segmentc
dc.w 14,38
.name |0
.name |1
dc.b 1,0        * Code-Segment
dc.l 0
pc_stand newequ 0
endmacro


* .SEGMENTD Segmentname, Groupname, Länge

macro segmentd
dc.w 14,38
.name |0
.name |1
dc.b 0,0        * Dcl-Segment
dc.l |2
endmacro

*********************************************************

tab     equ $09
cr      equ $0d
lf      equ $0a

lese       equ 3
motoff     equ 5
jresponse  equ 6
schreibe   equ 7
uppercas   equ 11
wrint      equ 13
close      equ 14
create     equ 16
erasefile  equ 17
fillfcb    equ 18
```

```
open       equ 19
readrec    equ 20
setdta     equ 22
writerec   equ 23
getversi   equ 24
getparm0   equ 25
jbell      equ 27
jbeep      equ 28
jerrnoise  equ 29
jsound     equ 30
jinport    equ 31
joutport   equ 32
wrtcmd     equ 36
gtretcod   equ 37
stretcod   equ 38
getparm1   equ 42
getparm2   equ 43
getparm3   equ 46
getparm4   equ 47
floppy     equ 51
jdrivecod  equ 52
jgetdrive  equ 53
jsetdrive  equ 54
getuhr     equ 65
jfileinfo  equ 71
jdiskinfo  equ 72
wrlint     equ 73
jlineedit  equ 77
getcpu     equ 78
jchmode    equ 83
jgetdpath  equ 85
asctonum   equ 88
numtoasc   equ 89
jgethdisk  equ 90
setrec     equ 93

saved   equ 15*4+4      * Anzahl gesaveter Bytes (incl. Ruecksprungadr.)
rsaved  equ 4

********************  MACROS  ************************

MACRO save
  movem.l d0-d7/a0-a6,-(a7)
ENDMACRO

MACRO rts0
  movem.l (a7)+,d0-d7/a0-a6
  rts
ENDMACRO

MACRO rtsq
  movem.l (a7)+,d0-d7/a0-a6
  move.l (a7),|0(a7)
  addq #|0,a7
  rts
ENDMACRO

MACRO rts
  movem.l (a7)+,d0-d7/a0-a6
  move.l (a7),|0(a7)
  adda #|0,a7
  rts
ENDMACRO

MACRO frtsq
  move.l (a7),|0(a7)
  addq #|0,a7
  rts
ENDMACRO

MACRO pushstring
  lea |0,a0
  suba #82,a7
```

```
  movea.l a7,a1
  clr d0
  move.b (a0),d0
  _ps||:
    move.b (a0)+,(a1)+
    dbra d0,_ps||
ENDMACRO

MACRO grundq
  moveq #!|0,d7
  trap #1
ENDMACRO

MACRO grund
  move #!|0,d7
  trap #1
ENDMACRO

MACRO jados
  moveq #|0,d7
  trap #6
ENDMACRO



MACRO real
  move.l #|3,-(a7)
  move.l #(|0)*$80000000+(|1)*$00100000+|2,-(a7)
ENDMACRO

bias    equ $3ff        * Offset für Exponent

_int0   equ saved+4
_int1   equ saved+2     * 1. Integer
_int2   equ saved       * 2. Integer

_var1   equ saved+4
_var2   equ saved



******************  ENDE MACROS  ********************

BEGIN

_pi0    equ saved+24    * alter Stack
_pi1b   equ saved+20    * Adresse __HIHEAP
_pi1a   equ saved+18    * Stackgroesse in KByte
_pi1ab  equ saved+16    * Heapgroesse in Kbyte
_pi1    equ saved+12    * Adresse _LOW_HEAP
_pi2    equ saved+8     * Adresse __HEAP
_pi3    equ saved+4     * Adresse _INPUT
_pi4    equ saved       * Adresse _OUTPUT

.modul .PASCAL-LIB
.segmentc INIT-CODE,CODE-GROUP
.segmentc CODE,CODE-GROUP
.segmentd _DATA,DATA-GROUP,0
.segmentc CODE,CODE-GROUP

.globproc _INITPAS
.globproc _PASINIT
.globproc _PASEXIT
.globproc CALLDUMP
.globproc cwrite
.globproc cread
.globproc clook
.globproc clook2
.globproc _CLOSE
.globproc _ERASE
* .globproc _REWRIT1
.globproc _REWRIT2
* .globproc _RESET1
```

```
.globproc _RESET2
.globproc APPEND
.globproc _NEW
.globproc _DISPOSE
.globproc _HALT

* .extproc numsign
* .extproc iread
* .extproc irdfehler
.extproc _convstr

.code

_initpas:
lea _stacksave(pc),a1
lea 4(a7),a0
move.l a0,(a1)
lea _chksave(pc),a1
lea tab_excep(pc),a0
_ip2:
  tst.w (a0)              * Tabellenende erreicht?
  beq.s _ip9
  movea.w (a0)+,a2        * Adresse Vektor
  move.l (a2),(a1)+       * alten Vektor retten
  lea tab_excep(pc),a3
  adda.w (a0)+,a3         * neue Zieladresse
  move.l a3,(a2)          * neuen Vektor speichern
  bra.s _ip2
_ip9:
suba.l a6,a6
rts

_pasexit:
lea _chksave(pc),a1
lea tab_excep(pc),a0
_pe2:
  tst.w (a0)
  beq.s _pe9
  movea.w (a0)+,a2
  move.l (a1)+,(a2)
  addq #2,a0
  bra.s _pe2
_pe9:
rts

tab_excep:
dc.w $0c,ex_addr-tab_excep
dc.w $10,ex_ill-tab_excep
dc.w $14,ex_div0-tab_excep
dc.w $18,ex_chk-tab_excep
dc.w $1c,ex_trapcc-tab_excep
dc.w $28,ex_aemul-tab_excep
dc.w $2c,ex_femul-tab_excep
dc.w 0,0

n_excep equ (*-tab_excep)/4-1


_pasinit:
.save
* lea _pi0(a7),a0
* lea _stacksave(pc),a1
* move.l a0,(a1)

  movea.l _pi1b(a7),a0    * Hiheap
  move _pi1a(a7),d0       * Stackgroesse
  mulu #1024,d0
* .jados wrlint
  move.l a7,d1
  sub.l d0,d1
  move _pi1ab(a7),d0      * Heapgroesse
  mulu #1024,d0
* .jados wrlint
```

```
  add.l _pi1(a7),d0
  cmp.l d0,d1
  blt _errmemsmall
  move.l d1,(a0)          * Hiheap fertig
movea.l _pi2(a7),a0     * Heap einrichten
move.l _pi1(a7),(a0)
movea.l _pi3(a7),a4     * RESET(INPUT,'CON:')
move.l a4,-(a7)
.pushstring _txtcon(pc)
bsr _reset2
movea.l _pi4(a7),a4     * REWRITE(OUTPUT,'CRT:')
move.l a4,-(a7)
.pushstring _txtcrt(pc)
bsr _rewrit2
.rts 4*4+2+2+4

_errmemsmall:
lea _txtmemsmall(pc),a0
.jados schreibe
bra _halt

_txtcon: dc.b 4,'CON:'
_txtcrt: dc.b 4,'CRT:'
_txtmemsmall: dc.b 'Speicher zu klein',0
_txtexit: dc.b cr,lf,'ENDE',cr,lf,0
ds 0

calldump:               * Ausgabe der bisherigen Unterprogrammaufrufe
.save
.grundq crlf
cmpa.l #0,a6
beq.s _cd9
_cd2:
  movea.l -6(a6),a0
  bsr _pri_name
  bcs.s cdsterr
  move.b #' ',d0
  .grundq co2
  clr.l d0
  move.w -2(a6),d0
  lea puffer(pc),a0
  .grundq print8d
  lea puffer(pc),a0
  bsr _pri_str
  .grundq crlf
  tst.l (a6)
  beq.s _cd9
  cmpa.l (a6),a6
  bhs.s cdsterr
  movea.l (a6),a6
  bra.s _cd2
_cd9:
.rts0

cdsterr:
lea tx_sterr(pc),a0
bsr _pri_str
.rts0

_pri_name:
moveq #9,d1
_pn2:
  move.b (a0)+,d0
  cmp.b #$20,d0
  bcs.s _pn9
  cmp.b #$7e,d0
  bcs.s _pn4
    or #1,ccr
    bra.s _pn9
  _pn4:
  .grundq co2
  dbra d1,_pn2
and #$fe,ccr
```

```
_pn9:
rts

_pri_str:
  move.b (a0)+,d0
  beq.s _ps9
  .grundq co2
  bra.s _pri_str
_ps9:
rts

MACRO excep
ex_|0:
lea tx_|0(pc),a1
bra.s ex_ex
ENDMACRO

.excep addr
.excep div0
.excep chk
.excep trapcc
.excep ill
.excep aemul
.excep femul
nop

ex_ex:
lea tx_lzerr(pc),a0
bsr _pri_str
movea.l a1,a0
bsr _pri_str
move.w __debug(a5),d1
beq.s ex_ex1
  lea tx_zl(pc),a0
  bsr _pri_str
  move d1,d0
  movem.l a6,-(a7)
  .jados wrint
  movem.l (a7)+,a6
ex_ex1:
lea tx_wahl(pc),a0
bsr _pri_str
.grundq ci
cmp.b #'c',d0
beq.s _exex2
cmp.b #'C',d0
bne.s _exex9
_exex2:
  bsr calldump
_exex9:
bra _halt

tx_lzerr:
dc.b cr,lf
dc.b 'Laufzeitfehler : ',0
tx_zl:
dc.b '     Code ',0
tx_sterr:
dc.b cr,lf,'Fehlerhafter Stackaufbau.',cr,lf,0
tx_addr:
dc.b 'Address-Fehler',0
tx_div0:
dc.b 'Division durch 0',0
tx_chk:
dc.b 'Index-Überschreitung',0
tx_trapcc:
dc.b 'Überlauf',0
tx_ill:
dc.b 'Ungültiger Befehl',0
tx_aemul:
dc.b 'Line A Emulator',0
tx_femul:
dc.b 'Line F Emulator',0
```

```
tx_wahl:
dc.b cr,lf
dc.b 'C = Calldump',cr,lf
dc.b 'E = Ende',cr,lf
dc.b 0
ds 0

_stacksave: ds.l 1
_chksave:   ds.l n_excep    * Platz für Vektoren
puffer:     ds.b 82



cwrite:
move.b 4(a4),d6
ble _rwfehler
move 6(a4),d7
lea _wrttab(pc),a6
adda.w 0(a6,d7),a6
jmp (a6)

_wrttab:
dc.w _wrtflo-_wrttab
dc.w _wrtcon-_wrttab
dc.w _wrtcrt-_wrttab
dc.w _wrtlst-_wrttab
dc.w _wrtnil-_wrttab
dc.w _wrtser-_wrttab

_wrtcrt:
.grundq co2

_wrtnil:
rts

_wrtlst:
.grundq lo
rts

_wrtser:
.grundq so
rts

_wrtcas:
.grundq po
rts

_wrtflo:
movem.l d1-d2/a0,-(a7)
movea.l (a4),a2
move.w (a2),d1
move.b d0,50(a2,d1)     * in Puffer schreiben
addq #1,d1
cmp #1024,d1    * Pufferende erreicht?
blt _wrtflo9
  clr d1        * wieder auf Anfang setzen
  lea 2(a2),a1  * FCB
  lea 50(a2),a0 * Speichertransferadresse
  .jados setdta
  .jados writerec
  tst.b d0
  bne _rwfehler
_wrtflo9:
clr.b 50(a2,d1)   * mit Null abschliessen
move.w d1,(a2)
movem.l (a7)+,d1-d2/a0
rts

_wrtcon:
_rwfehler:
lea _txtrwerr(pc),a0
.jados schreibe
bra _halt
```

```
_txtrwerr:
dc.b 'Schreib/Lese-Fehler',cr,lf,0
ds 0

*  REWRITE, RESET und CLOSE
*****************************************

* Datei-Variablen-Aufbau:
* Jede Datei-Variable besteht aus 8 Bytes.
*
* Byte   Bedeutung
*-----------------------------------------*
*   0    1 Long   Zeiger auf Puffer
*   4    1 Byte   -1 = closed  1 = zum Schreiben geöffnet
*   6    1 Wort   0=Datei  sonst: 2*Geraetenummer
*-----------------------------------------*

* Puffer-Format für Dateien
* jeweils 1074 Byte, wird auf dem Heap angelegt.
*
* Byte   Bedeutung
*-----------------------------------------*
*   0    1 Wort   0..1023 aktuelle Position im Puffer
*   2   48 Byte   FCB
*  50 1024 Byte   Puffer
*-----------------------------------------*

* Puffer-Format für CON:
* jeweils 2+CIBUFSIZ Byte, wird auf dem Heap angelegt.
*
* Byte   Bedeutung
*-----------------------------------------*
*   0         1 Wort   0.. aktuelle Position im Puffer
*   2  CIBUFSIZ Byte   Puffer
*-----------------------------------------*

cibufsiz equ 82


_res1   equ saved+82    * File
_res2   equ saved       * String (Filename)

append:
.save
move #%100000010,d3
bra.s _rst1

_rewrit2:
.save
moveq #%000000010,d3
bra.s _rst1

_reset2:
.save
moveq #%000000001,d3

_rst1:
movea.l _res1(a7),a4    * Zeiger auf Datei
lea _res2(a7),a0        * Zeiger auf Name
bsr _convstr            * in ASCIIZ umwandeln
.jados uppercas,        * und Groábuchstaben
bsr _getdevice
and.b d3,d1             * Schreib- bzw. Lesezugriff erlaubt?
beq _rstfehler
tst d0
beq _rst5

cmp #1,d0       * CON: ?
bne _rst4
  move.l a4,-(a7)
  move #2+cibufsiz,-(a7)
```

```
  bsr _new
  movea.l (a4),a2
  move #cibufsiz,(a2)

_rst4:
  add d0,d0
  move d0,6(a4)          * Fileart
  move.b d3,4(a4)        * Read/Write
  bra _rst9
_rst5:
  move.l a4,-(a7)
  move #2+48+1024,-(a7)   * Speicheranforderung fuer Pointer, FCB und
  Puffer
  bsr _new
  movea.l (a4),a2
  lea 2(a2),a1           * FCB
  .jados fillfcb
  tst.b d0
  bne _rstfehler
  clr 6(a4)
  cmp.b #%10,d3
  bne.s _rst8
    move.b #1,4(a4)      * Write
    btst #8,d3           * append?
    beq.s _rst7
      .jados open
      tst.b d0
      bne.s _rst7
      move.w 26(a1),d1
      subq #1,d1
      .jados setrec,     * letzten Sektor
      tst.b d0
      bne _rstfehler
      lea 50(a2),a0
      .jados setdta
      .jados readrec
      tst.b d0
      bne _rstfehler
      .jados setrec
      move #-1,(a2)
      _rst6:             * Endekennung suchen
        addq #1,(a2)
        tst.b (a0)+
        beq _rst9
        cmp #1023,(a2)
        beq _rstfehler
        bra.s _rst6
    _rst7:
    .jados create
    tst.b d0
    bne _rstfehler
    clr (a2)
    clr.b 50(a2)
    bra.s _rst9
  _rst8:
    clr.b 4(a4)          * Read
    .jados open
    tst.b d0
    bne _rstfehler
    move #1024,(a2)
_rst9:
.rts 82+4

_rstfehler:
lea _txtrsterr(pc),a0
.jados schreibe
bra _halt

_txtrsterr:
dc.b 'Fehler beim Öffnen einer Datei',cr,lf,0
ds 0

_getdevice:     * D0: Devicenummer, D1: Read/Write
```

```
movea.l a0,a1   * String nach erstem Wort beenden
_getdev2:
  move.b (a1)+,d0
  beq.s _getdev3
  cmp.b #' ',d0
  bne.s _getdev2
clr.b -1(a1)
_getdev3:
clr d0
tst.b 4(a0)
bne _getdev9    * Datei
move.l (a0),d1  * Geraetename
lea _devicetab(pc),a1
moveq #1,d0
_getdev4:
  tst.b (a1)
  bne.s _getdev5
    clr d0          * Kennzeichen für Datei
    bra _getdev9    * Datei
  _getdev5:
  cmp.l (a1),d1
  beq _getdev6      * Device gefunden
  addq #6,a1
  addq #1,d0
  bra.s _getdev4
_getdev6:
move.b 4(a1),d1
rts
_getdev9:
moveq #2+1,d1       * Datei: sowohl lesen als auch schreiben erlaubt
rts


_devicetab:
*             WR (schreib-/lesbar)
*--------------------------------
dc.b 'CON:',%01,0
dc.b 'CRT:',%10,0
dc.b 'LST:',%10,0
dc.b 'NIL:',%10,0
dc.b 'SER:',%11,0
dc.b 0,0
dc.b 'CAS:',%11,0
dc.b 'USR:',%11,0
dc.b 'AUX:',%11,0
dc.b 'RAM:',%11,0
*----------------

_cl1    equ saved   * File

_erase:
.save
movea.l _cl1(a7),a4
tst 6(a4)
bne _erase8
tst.b 4(a4)
bmi _erase8
  movea.l (a4),a2
  lea 2(a2),a1
  .jados erasefile
  tst.b d0
  bne _diskerr
  bra _close8
_erase8:
.rtsq 4

_close:
.save
movea.l _cl1(a7),a4
tst 6(a4)
bne _close8             * nur bei Dateien was tun
tst.b 4(a4)
beq _close7             * bei Schreibzugriff Puffer schreiben
```

```
blt _clfehler
  movea.l (a4),a2       * bei Schreibzugriff Puffer schreiben
  lea 2(a2),a1          * FCB
  lea 50(a2),a0         * Speichertransferadresse
  .jados setdta
  .jados writerec
_close7:
  movea.l (a4),a2
  lea 2(a2),a1
  .jados close
  move.l a4,-(a7)       * Puffer vom Heap entfernen
  move #2+48+1024,-(a7)
  bsr _dispose
* lea txtc(pc),a0
* .jados schreibe
_close8:
move.b #-1,4(a4)        * Datei geschlossen
.rtsq 4

_clfehler:
lea _txtclerr(pc),a0
.jados schreibe
bra _halt

_diskerr:
lea _txtderr(pc),a0
.jados schreibe
bra _halt

txtc:
dc.b 'Datei geschlossen',cr,lf,0
_txtclerr:
dc.b 'Fehler bei CLOSE: Datei ist nicht geöffnet',cr,lf,0
_txtderr:
dc.b 'Fehler bei Diskettenzugriff',cr,lf,0
ds 0



_rdlf:
bsr _rd
cmp.b #lf,d0
bne.s _rdlf9
  moveq #' ',d0
_rdlf9:
rts

clook2:
bsr _rd
bra.s clook3

clook:
bsr _rdlf
clook3:
tst d6
beq.s clookflo
  move.b d0,(a2)
  st 1(a2)
  rts
clookflo:
  subq.w #1,(a2)   * eins zurueckstellen
  rts

cread:
bsr _rdlf
cmp.b #cr,d0
bne.s cread9
  bsr _rdlf     * LF auch lesen
  moveq #' ',d0
cread9:
tst d6
beq.s cread99
  clr.b 1(a2)   * bei Geraeten auf ungueltig stellen
```

```
cread99:
rts


_rd:
move 6(a4),d6
beq _rdflo

lea insave(pc,d6),a2
move.b (a2),d0
tst.b 1(a2)
beq.s _rd2
  rts
_rd2:
lea _rdtab(pc),a6
adda.w 0(a6,d6),a6
jmp (a6)

insave:
dc.b 0,0
dc.b 0,0
dc.b 0,0
dc.b 0,0
dc.b 0,0

_rdtab:
dc.w _rdflo-_rdtab
dc.w _rdcon-_rdtab
dc.w _rdcrt-_rdtab
dc.w _rdlst-_rdtab
dc.w _rdnil-_rdtab
dc.w _rdser-_rdtab

_rdflo:
movem.l d1/a0-a1,-(a7)
movea.l (a4),a2
move (a2),d1
cmp #1024,d1
blt _rdflo2
  lea 2(a2),a1
  lea 50(a2),a0
  .jados setdta
  .jados readrec
  clr d1
_rdflo2:
move.b 50(a2,d1),d0
addq #1,d1
move d1,(a2)
movem.l (a7)+,d1/a0-a1
rts

_rdcon:
movem.l d1-d2/a1-a2,-(a7)
movea.l (a4),a2
move (a2),d1
cmp #cibufsiz,d1
blt _rdcon2
  lea 2(a2),a1
  clr.b (a1)            * Puffer vorlöschen
  .grund getcurxy
  move #cibufsiz-3,d2
  sub d1,d2             * Nur bis zum Zeilenende
* .jados lese
  .jados jlineedit
  move d0,d2
  .grundq crlf
  clr d1                * Am Pufferanfang beginnen

  cmp.b #3,d2   * Fehlercode von LINEEDIT
* cmp.b #$1b,d2 * Fehlercode LESE

  beq _halt     * gegebenenfalls Programm anhalten (bei ^C)
_rdcon2:
```

```
move.b 2(a2,d1),d0
bne.s _rdcon3           * Zeilenende erreicht?
  moveq #cr,d0
  move #cibufsiz-2,d1   * LF muá als nächstes gelesen werden
  move.b #lf,3(a2,d1)   * also an letzter Lesepos. eintragen
_rdcon3:
addq #1,d1
move d1,(a2)
movem.l (a7)+,d1-d2/a1-a2
rts

_rdser:
.grundq si
rts

_rdcas:
.grundq ri
rts

_rdnil:
clr d0
rts

_rdcrt:
_rdlst:
bra _halt



*****************************************

_halt:
movea.l _stacksave(pc),a7
rts

* NEW und DISPOSE
*****************************************

__HEAP      EQU  -32000
__HIHEAP    EQU  -31996
__DEBUG     EQU  -31992


_new1   equ saved+2     * Adresse  Pointer
_new2   equ saved       * Laenge

_new:           * Achtung!! Absolute PRIMITIV-Version !!!
.save
movea.l _new1(a7),a0
move.l __heap(a5),d1
move.l d1,(a0)          * Zeiger auf momentanen Heap
clr.l d0
move _new2(a7),d0
add.l d0,d1
move.l d1,__heap(a5)    * Heap weiterstellen
  cmp.l __hiheap(a5),d1
  bgt.s _newerr
.rtsq 4+2


_dispose:       * tut derzeit einfach nichts
.save
movea.l _new1(a7),a0
clr.l (a0)      * auf NIL setzen
.rtsq 4+2

_newerr:
lea _txtnewerr(pc),a0
.jados schreibe
bra _halt

_txtnewerr:
dc.b 'HEAP šberlauf',cr,lf,0
```

```
ds 0


.endcode
.endmodul

*******************************************************************************
*****

END
BEGIN


* READ und READLN
*****************************************

.modul .Read
.segmentc CODE,CODE-GROUP

.globproc _READLN
.globproc _IREAD
.globproc _CREAD
.globproc _SREAD
* .globproc _ACREAD
.globproc numsign
.globproc iread
.globproc irdfehler

.extproc cread
.extproc clook
.extproc clook2
.extproc _HALT


.code


_ir1    equ saved+4     * File
_ir2    equ saved       * VAR Integer

_iread:
.save
movea.l _ir1(a7),a4
movea.l _ir2(a7),a3
bsr iread
move.w d0,(a3)
.rtsq 4

_cread:
.save
movea.l _ir1(a7),a4
movea.l _ir2(a7),a3
bsr cread
move.b d0,(a3)
.rtsq 4

_sread:
.save
movea.l _ir1(a7),a4
movea.l _ir2(a7),a3
clr d1
addq #1,a3
_sread2:
  bsr clook
  cmp.b #cr,d0
  beq.s _sread3
  bsr cread
  move.b d0,(a3)+
  addq #1,d1
  bra.s _sread2
_sread3:
movea.l _ir2(a7),a3
move.b d1,(a3)
```

```
.rtsq 4

_rdln1  equ saved       * File

_readln:
.save
movea.l _rdln1(a7),a4
_readln2:
  bsr clook
  cmp.b #cr,d0
  beq.s _readln3
  bsr cread
  bra.s _readln2
_readln3:
bsr cread
bsr clook2      * evtl auch LF weglesen
cmp.b #lf,d0
bne.s _readln9
  bsr cread
_readln9:
.rtsq 4

numsign:
bsr cread
nsign1:
  cmp.b #' ',d0
  bne.s nsign2
  bsr cread
  bra.s nsign1
nsign2:
clr d3
cmp.b #'+',d0
beq.s nsign3
cmp.b #'-',d0
seq d3          * Vorzeichen merken
bne.s nsign4
  nsign3:
  bsr cread
nsign4:
rts

iread:
clr.l d1
bsr numsign
iread2:
  cmp.b #'9',d0
  bhi irdfehler
  sub.b #'0',d0
  blo irdfehler
  ext.w d0
  ext.l d0
  add.l d1,d1   * mal 2
  move.l d1,d2
  lsl.l #2,d1   * mal 8
  add.l d2,d1   * mal 10
  add.l d0,d1
  bsr clook
  cmp.b #' ',d0
  beq.s iread9
  cmp.b #cr,d0
  beq.s iread9
  bsr cread
  bra.s iread2
iread9:
move.l d1,d0
tst.b d3
beq.s iread99
  neg.l d0
iread99:
rts

irdfehler:
lea _txtirderr(pc),a0
```

```
.jados schreibe
bra _halt

_txtirderr:
dc.b 'Fehler bei numerischer Eingabe',cr,lf,0
ds 0


.endcode
.endmodul

*************************************************************************

END
BEGIN


* EOF und EOLN
*****************************************


.modul .EOF/EOLN
.segmentc CODE,CODE-GROUP


.globproc _EOF
.globproc _EOLN

.extproc clook

.code


_eox    equ saved+4 * Funktionswert
_eo1    equ saved   * File

_eof:
.save
movea.l _eo1(a7),a4
bsr clook
cmp.b #0,d0
_eo9:
seq d0
and #1,d0           * booleschen Wert erzeugen
move.b d0,_eox(a7)  * Funktionswert setzen
.rtsq 4


_eoln:
.save
movea.l _eo1(a7),a4
bsr clook
cmp.b #cr,d0
bra.s _eo9


.endcode
.endmodul

*************************************************************************

END
BEGIN


.modul .Read-REAL
.segmentc CODE,CODE-GROUP


.globproc _RREAD

.extproc _FADD
.extproc _FMUL
```

```
.extproc _FLOAT
.extproc _FNEG
.extproc IPOWER
.extproc _FEXPONENT
.extproc clook
.extproc cread
.extproc numsign
.extproc iread
.extproc irdfehler


.code


_rread:
.save
movea.l saved+4(a7),a4
movea.l saved(a7),a3
bsr rread
move.l d0,(a3)
move.l d1,4(a3)
* fmove.d fp0,(a3)
.rtsq 4


rread:
  .real 0,0,0,0
* fmove.s #0.0,fp0
bsr numsign
move d3,d4
rread2:
  cmp.b #'9',d0
  bhi rrdfehler
  sub.b #'0',d0
  blo rrdfehler
  .real 0,3+bias,$40000,0
  bsr _fmul
* fmul.s #10,fp0
  subq #8,a7    * Platz für REAL
  ext.w d0
  move.w d0,-(a7)
  bsr _float
  bsr _fadd
* fadd.b d0,fp0
  bsr clook
  cmp.b #' ',d0
  beq rread9
  cmp.b #cr,d0
  beq rread9
  bsr cread
  cmp.b #'e',d0
  beq.s rread5
  cmp.b #'E',d0
  beq.s rread5
  cmp.b #'.',d0
  bne.s rread2
clr d1
rread4:
  bsr clook
  cmp.b #' ',d0
  beq.s rread7
  cmp.b #cr,d0
  beq.s rread7
  bsr cread
  cmp.b #'e',d0
  beq.s rread6
  cmp.b #'E',d0
  beq.s rread6
  cmp.b #'9',d0
  bhi rrdfehler
  sub.b #'0',d0
  blo rrdfehler
  .real 0,3+bias,$40000,0
```

```
  bsr _fmul
* fmul.s #10,fp0
  subq #8,a7    * Platz für REAL
  ext.w d0
  move.w d0,-(a7)
  bsr _float
  bsr _fadd
* fadd.b d0,fp0
  subq #1,d1
  bra rread4
rread5:
clr d1
rread6:
move d1,-(a7)
bsr iread
move (a7)+,d1
add d0,d1
rread7:
  subq #8,a7      * Platz für REAL
  .real 0,3+bias,$40000,0
  move d1,-(a7)
  bsr ipower
* ftentox.w d1,fp1
* fmove.d fp1,-(a7)
bsr _fmul
* fmul fp1,fp0
rread9:
tst.b d4
beq.s rread99
  bsr _fneg
rread99:
move.l (a7)+,d0
move.l (a7)+,d1
rts

rrdfehler:
addq #8,a7      * REAL vom Stack nehmen
bra irdfehler


.endcode
.endmodul

*****************************************************************************

END
BEGIN

*  WRITE und WRITELN
**************************

.modul .Write
.segmentc CODE,CODE-GROUP

.globproc _WRITELN
.globproc _IWRITE
.globproc _CWRITE
.globproc _SWRITE
.globproc _ACWRITE
.globproc swrite
.globproc spaces

.extproc cwrite


.code

_wrtln1   equ saved


_writeln:
.save
movea.l _wrtln1(a7),a4      * File stets in A4
```

```
moveq #cr,d0
bsr cwrite
moveq #lf,d0
bsr cwrite
.rtsq 4



puffer: ds.b 12         * fuer Dezimalzahlen



_iw1    equ saved+4     * File
_iw2    equ saved+2     * Integer
_iw3    equ saved       * Option

_iwrite:        * WRITE(Integer)
.save
movea.l _iw1(a7),a4
move _iw2(a7),d0
ext.l d0
lea puffer(pc),a0
.grundq printv8d
move.l a0,d2
lea puffer(pc),a0
sub.l a0,d2
move _iw3(a7),d1
bsr swrite
.rtsq 2+2


_cw1    equ saved+4     * File
_cw2    equ saved+2     * Char
_cw3    equ saved       * Option

_cwrite:        * WRITE(Char)
.save
movea.l _cw1(a7),a4
move _cw3(a7),d1
subq #1,d1
bsr spaces
move.b _cw2(a7),d0
bsr cwrite
.rtsq 2+2


_sw1    equ saved+6     * File
_sw2    equ saved+2     * Stringanfang
_sw3    equ saved       * Option

_swrite:        * WRITE(String)
.save
movea.l _sw1(a7),a4
movea.l _sw2(a7),a0
move _sw3(a7),d1
clr d2
move.b (a0)+,d2    * Stringlänge
bsr swrite
.rtsq 2+4


_acw1   equ saved+8     * File
_acw2   equ saved+4     * Array
_acw3   equ saved+2     * Laenge
_acw4   equ saved       * option

_acwrite:       * WRITE(ARRAY of Char)
.save
movea.l _acw1(a7),a4
movea.l _acw2(a7),a0
move _acw3(a7),d2
move _acw4(a7),d1
```

```
bsr swrite
.rtsq 4+2+2


swrite:
sub d2,d1
bsr spaces
subq #1,d2
bmi.s swrite4
swrite2:
  move.b (a0)+,d0
  bsr cwrite
  dbra d2,swrite2
swrite4:
rts

spaces:
subq #1,d1
bmi.s spaces9
  spaces2:
    moveq #' ',d0
    bsr cwrite
    dbra d1,spaces2
spaces9:
rts



.endcode
.endmodul

*******************************************************************************
*****


END
BEGIN


.modul .Write-REAL
.segmentc CODE,CODE-GROUP

.globproc _RWRITE

.extproc _FADD
.extproc _FNEG
.extproc _FMUL
.extproc IPOWER
.extproc cwrite
.extproc spaces
.extproc swrite


.code


puffer: ds.b 82

_rw1    equ saved+12    * File
_rw2    equ saved+4     * Real
_rw3    equ saved+2     * Option-Total
_rw4    equ saved       * Option-Frac


_rwrite:        * WRITE(Real)
.save
movea.l _rw1(a7),a4

fmove.d _rw2(a7),fp0
move _rw3(a7),d0
move _rw4(a7),d1
bpl _fwrite             * Fixedpunktdarstellung
subq #7,d0              * Dezimalpunkt, Vorzeichen und Exponent
```

```
bgt.s _rw_1
  moveq #1,d0           * mindestens eine Stelle anzeigen
_rw_1:
cmp #16,d0              * aber höchstens 16 Stellen
ble.s _rw_1a
  moveq #16,d0
_rw_1a:
lea puffer(pc),a0

ftst fp0

fbolt _rw_2
  move.b #' ',(a0)+
_rw_2:
.grundq printfp0
lea puffer(pc),a0       * Jetzt noch hintere Stellen wegwerfen
lea 2(a0,d0),a0
movea.l a0,a1           * 'e' suchen
_rw_3:
  cmp.b #'e',(a1)+
  bne.s _rw_3
move.b #'e',(a0)+       * a1 zeigt jetzt hinter 'e'
cmp.b #'-',(a1)
beq.s _rw_4
  move.b #'+',(a0)+     * Bei pos. Exponenten '+' einfügen
_rw_4:
  move.b (a1)+,(a0)+    * Heranschieben
  bne.s _rw_4
subq #1,a0              * zeigt jetzt auf Ende-Null
move.l a0,d2
lea puffer(pc),a0
sub.l a0,d2
move _rw3(a7),d1
bsr swrite
.rts 12

_fwrite:
* fmove.d fp0,-(a7)       * zum Ausprobieren _____________________
lea puffer(pc),a0
clr d3
* tst.b (a7)            * Vorzeichen testen
* bpl.s _fw_1

  ftst fp0               * Vorzeichen
  fboge _fw_1
  moveq #1,d3
* bsr _fneg

  fneg fp0
_fw_1:

fneg.w d1,fp1
ftentox fp1
fmul.s #0.5,fp1
fadd fp1,fp0            * geeignet runden
flog10 fp0,fp1
fintrz fp1
fmove.w fp1,d2
tst d2
bpl.s _fw_2
  clr d2
_fw_2:                  * Jetzt Anzahl Vorkommastellen-1 in d2
movem d1,-(a7)           * sichern
 add d2,d1
 addq #2,d1             * Dezimalpunkt und Korrektur Vorkomma
 tst.b d3
 beq.s _fw_2a
   addq #1,d1
 _fw_2a:
 neg d1
 add d0,d1
 bsr spaces
 tst.b d3
```

```
 beq.s _fw_2b
   move.b #'-',d0
   bsr cwrite
 _fw_2b:
movem (a7)+,d1

ftentox.w d2,fp1
fdiv fp1,fp0
_fw_3:
  bsr onedigit
  dbra d2,_fw_3
move.b #'.',d0
bsr cwrite
subq #1,d1
bmi.s _fw_8
_fw_4:
  bsr onedigit
  dbra d1,_fw_4
_fw_8:
.rts 12

onedigit:

  fintrz fp0,fp1
  fmove.b fp1,d0
  fsub fp1,fp0
  add.b #'0',d0
  bsr cwrite

  fmul.s #10,fp0
rts



.endcode
.endmodul

*******************************************************************************
*****


END
BEGIN

.modul .String-ASCIIZ
.segmentc CODE,CODE-GROUP

.globproc _convstr

.code

_convstr:
lea zpuffer(pc),a1
clr d0
move.b (a0)+,d0
subq #1,d0
bmi.s _conv2
_conv3:
  move.b (a0)+,(a1)+
  dbra d0,_conv3
_conv2:
clr.b (a1)
lea zpuffer(pc),a0
rts

zpuffer: ds.b 82

.endcode
.endmodul

*******************************************************************************
*****
```

```
END
BEGIN

.modul .Gleitkomma
.segmentc CODE,CODE-GROUP


macro f_func
  _f|0:
  .save
  bsr load1f
  bsr f|0
  bra store1f
endmacro


.globproc _FLOAT
.globproc _FLOAT2
.globproc _FTRUNC
.globproc _FROUND
.globproc _FNEG
.globproc _FADD
.globproc _FSUB
.globproc _FMUL
.globproc _FDIV
.globproc _FCMP
.globproc _FABS
.globproc _FSQR
.globproc _FINT
.globproc _FFRAC
.globproc _FEXPONENT
.globproc IPOWER

.globproc _FSQRT
.globproc _FEXP

.code


load1f:
lea saved+4(a7),a0      * zusätzliche Rücksprungadresse

loadf:
move.l (a0),d0
move.l d0,d1
move.l d0,d7
and.l #$80000000,d1
swap d0
lsr #4,d0
and.l #$7ff,d0     * Exponent steht jetzt in d0
or.l d1,d0         * VZ Bit dazu
and.l #$000fffff,d7     * Mantisse abspalten
tst.w d0
beq.s l1f3
  or.l #$00100000,d7    * normalisiert
  bra.s l1f4
l1f3:
  addq.w #1,d0          * unnormalisiert
l1f4:
swap d7
ror.l #5,d7
move.w 4(a0),d1
lsr #5,d1
and.l #$07ff,d1
or.l d7,d1              * Mantisse komplett zusammengesetzt
sub #bias,d0            * Exponent jetzt als Zweierkomplement
rts


load2f:
lea saved+4(a7),a0
bsr loadf               * 2. Operand nach d3d4
```

```
move.l d0,d3
move.l d1,d4
lea saved+8+4(a7),a0
bra loadf               * 1. Operand nach d0d1


storef:
tst.l d1        * Null?
beq.s storef1
add #bias,d0
bgt.s storef2   * Underflow?
  storef1:
  and.l #$80000000,d0
  clr.l d1
  bra.s storef8
storef2:
move.l d0,d7
and.l #$80000000,d7
and.l #$7ff,d0
swap d0
lsl.l #4,d0
or.l d7,d0
move.l d1,d7
swap d7
rol.l #5,d7
and.l #$fffff,d7
or.l d7,d0              * VZ, Exp. und 20 Bit Mantisse
and.l #$7ff,d1
swap d1
lsl.l #5,d1
storef8:
move.l d0,(a0)
move.l d1,4(a0)
rts

store1f:
lea saved(a7),a0
bsr storef
.rts0

store2f:
lea saved+8(a7),a0
bsr storef
.rtsq 8

*** Vorzeichen: fneg und fabs

_FNEG:
eor.b #$80,rsaved(a7)   * nur Vorzeichen herumdrehen
rts

_FABS:
and.b #$7f,rsaved(a7)   * Vorzeichen löschen
rts

*** Umwandlung int -> real: float und float2

_FLOAT:
.save
move saved(a7),d0
ext.l d0
bsr float
lea saved+2(a7),a0
bsr storef
.rtsq 2

_FLOAT2:
.save
move.l saved+14(a7),saved+6(a7)
move.l saved+10(a7),saved+2(a7)
move saved(a7),d0
ext.l d0
bsr float
```

```
lea saved+10(a7),a0
bsr storef
.rtsq 2


_FTRUNC:
.save
bsr load1f
bsr ftrunc
move d1,saved+8(a7)
.rtsq 8

_FROUND:
.save
bsr load1f
bsr fround
move d1,saved+8(a7)
.rtsq 8

_FEXPONENT:
.save
bsr load1f
move d0,saved+8(a7)     * Exponent steht in d0.w
.rtsq 8

_FADD:
.save
bsr load2f
bsr fadd
bra store2f

_FSUB:
.save
bsr load2f
bsr fsub
bra store2f

_FMUL:
.save
bsr load2f
bsr fmul
bra store2f

_FDIV:
.save
bsr load2f
bsr fdiv
bra store2f

_FCMP:
.save
bsr load2f
_fcmp2:
bsr fcmp
.rts0

IPOWER:
.save
lea saved+2(a7),a0
bsr loadf
move saved(a7),d3
ext.l d3
bsr fipower
lea saved+10(a7),a0
bsr storef
.rts 10

.f_func SQR
.f_func SQRT
.f_func INT
.f_func FRAC
.f_func EXP
```

```
********************** Realisierung für internes Format

cmzero  equ $10000-bias


fzero:
clr.l d1
move.l #cmzero,d0
rts

sfzero:
clr.l d1
move #$10000-bias,d0
rts

sfinfty:
move.l #$80000000,d1
move.l #$ffff-bias,d0
rts

fnan:
move.l #$7fffffff,d1
move.l #$ffff-bias,d0
rts


*******************************************
* FLOAT   Wandlung nach REAL              *
* Argument in d0 Ergebnis in (d0,d1)      *
* Zerstörte Register:   keine             *
*******************************************
float:
move.l d0,d1
beq.s fzero
bpl.s fl12
  move.l #$8000001f,d0   * -2^31
  neg.l d1
  bra.s fl13
fl12:
  moveq #$1f,d0          * +2^31
fl13:
fl14:
  subq #1,d0
  lsl.l #1,d1
  bpl.s fl14
rts


*** Grundrechenarten


*****************************************
* FSUB  Subtraktion (d3,d4) von (d0,d1)   *
* FADD  Addition (d3,d4) zu (d0,d1)       *
* Zerstörte Register:   d3,d4,d7          *
*****************************************
fsub:
bchg #31,d3     * Vorzeichen rumdrehen
fadd:
cmp d3,d0
bgt.s fadd1
  exg d0,d3
  exg d1,d4
fadd1:
tst.l d1
beq.s addnull
move.l d0,d7
sub d3,d7
bvs.s retadd
cmp #32,d7
bgt.s retadd
lsr.l d7,d4
bcc.s fadd11
```

```
  addq.l #1,d4
fadd11:
eor.l d3,d7
bmi.s addpm
  add.l d4,d1
  bcc.s retadd
  roxr.l #1,d1
  bcc.s addpp1
    addq.l #1,d1
  addpp1:
  addq #1,d0
  retadd:
  rts
addpm:
  sub.l d4,d1
  bcc.s addpm1
    bchg #31,d0
    neg.l d1
  addpm1:
  bmi.s addpm3
  beq fzero
  addpm2:
    subq #1,d0
    bvs fzero
    lsl.l #1,d1
    bpl.s addpm2
  addpm3:
  rts
addnull:
move.l d3,d0
move.l d4,d1
rts


*********************************************
* FSQR  Quadrieren von (d0,d1)              *
* Zerstörte Register:   d3,d4,d6,d7         *
*********************************************
fsqr:
move.l d0,d3
move.l d1,d4
bra.s fmul


*********************************************
* FDIV  Division (d0,d1) durch (d3,d4)      *
* Zerstörte Register:   d3,d4,d7            *
*********************************************
fdiv:
movem.l d0-d1,-(a7)
move.l d3,d0
move.l d4,d1
bsr kehrwert
move.l d0,d3
move.l d1,d4
movem.l (a7)+,d0-d1
*********************************************
* FMUL   Multiplikation (d0,d1) mit (d3,d4) *
* Zerstörte Register:   d7                  *
*********************************************
fmul:
tst.l d1
beq fzero
tst.l d4
beq fzero

link a0,#-8
tst.l d3
bpl.s fmul2
  bchg #31,d0
fmul2:
add d3,d0
```

```
move d1,d7
mulu d4,d7
move.l d7,-4(a0)

move.l d1,d7
swap d7
swap d4
mulu d4,d7
swap d4
move.l d7,-8(a0)

move.l d1,d7
swap d7
mulu d4,d7
add.l d7,-6(a0)
bcc.s fmul31
  addq.w #1,-8(a0)
fmul31:

move.l d4,d7
swap d7
mulu d1,d7
add.l d7,-6(a0)
bcc.s fmul32
  addq.w #1,-8(a0)
fmul32:

move.l -8(a0),d1
bmi.s fmul5
  lsl -4(a0)
  roxl.l #1,d1
  bra.s fmul6
fmul5:
  addq.w #1,d0
fmul6:
tst.b -4(a0)    * runden
bpl.s fmul9
  addq.l #1,d1  * kann es hier einen šberlauf geben???
fmul9:
unlk a0
rts


*******************************************
* KEHRWERT  Kehrwert berechnen             *
*           Argument und Ergebnis in (d0,d1)*
* Zerstörte Register:    d3,d4,d6,d7       *
*******************************************
kehrwert:
tst.l d1
*  beq fdiv0
cmp.l #$80000000,d1
bne.s fkw1
  neg d0
  rts
fkw1:
link a0,#-16
  nenner   equ -8
  quotient equ -16
move.l d0,d6
bchg #31,d6
move.l d6,nenner(a0)
move.l d1,nenner+4(a0)
move.l d1,d7
swap d7
cmp #$8000,d7
bne.s fkw2
  move.l #$ffff0000,d1
  bra.s fkw4
fkw2:
  move.l #$80000000,d6
  divu d7,d6
  swap d6
```

```
  move.w #$4000,d6  * Symmetrisierung des Fehlers im Mittel
  move.l d6,d1
fkw4:
not d0          * d0 := -d0-1
* bra fkw9      * Iteration überspringen
move.l d0,quotient(a0)
move.l d1,quotient+4(a0)
move.l nenner(a0),d3
move.l nenner+4(a0),d4
bsr fmul
moveq #1,d3
move.l #$80000000,d4
bsr fadd
move.l quotient(a0),d3
move.l quotient+4(a0),d4
bsr fmul
fkw9:
unlk a0
rts


*******************************************
* FIPOWER  Potenzieren (d0,d1) mit d3     *
* Zerstörte Register:  d3,d4,d6,d7        *
*******************************************
fipower:
link a0,#-12
  merk    equ -8
  vz      equ -10
  zaehl   equ -12

tst.l d3
smi vz(a0)
bpl.s powint1
  neg.l d3

powint1:
move.l d0,merk(a0)
move.l d1,merk+4(a0)
clr.l d0                * 1.0
move.l #$80000000,d1
tst.l d3
beq powint5

moveq #31,d4
powint2a:
  subq #1,d4
  lsl.l #1,d3
  bpl.s powint2a
move d4,zaehl(a0)

powint2:
  lsl.l #1,d3
  bcc.s powint3
    move.l d3,-(a7)
    move.l merk(a0),d3
    move.l merk+4(a0),d4
    bsr fmul
    move.l (a7)+,d3
  powint3:
  subq #1,zaehl(a0)
  bmi.s powint4
  move.l d3,-(a7)
  bsr fsqr
  move.l (a7)+,d3
  bra.s powint2
powint4:
tst.b vz(a0)
beq.s powint5
  bsr kehrwert

powint5:
unlk a0
```

```
rts


*******************************************
* FINT  Berechnung des Ganzzahlanteils    *
*       Argument und Ergebnis in (d0,d1)  *
* Zerstörte Register:   d7                *
*******************************************
fint:
tst d0
blt fzero
moveq #31,d7
sub d0,d7
bcs.s fint9
lsr.l d7,d1
lsl.l d7,d1
fint9:
rts


*******************************************
* FFRAC   Berechnung des Nachkommateils   *
*         Argument und Ergebnis in (d0,d1)*
* Zerstörte Register:   d3,d4,d7          *
*******************************************
ffrac:
movem.l d0-d1,-(a7)
bsr fint
move.l d0,d3
move.l d1,d4
movem.l (a7)+,d0-d1
bra fsub


*******************************************
* FROUND  Rundung auf Integer             *
* Argument (d0,d1) Ergebnis d1            *
* Zerstörte Register:   d3,d4,d7          *
*******************************************
fround:
move.l d0,d3
move.w #$ffff,d3
move.l #$80000000,d4    * 0.5 mit gleichem Vorzeichen
bsr fadd
*******************************************
* FTRUNC  Abschneiden auf Integer         *
* Argument (d0,d1) Ergebnis d1            *
* Zerstörte Register:   d7                *
*******************************************
ftrunc:
tst.l d1
beq.s ftr9
tst d0
bge.s ftr2
  clr.l d1
  rts
ftr2:
moveq #31,d7
sub d0,d7
* bcs ftrerr
lsr.l d7,d1
tst.l d0
bpl.s make1
  neg.l d1
make1:
ftr9:
rts


*******************************************
* FCMP   Vergleich (d0,d1) mit (d3,d4)    *
* Ergebnis CC                             *
* Zerstörte Register:   keine             *
```

```
*******************************************
fcmp:
move.l d0,d7
eor.l d3,d7
bpl.s fcmp2
  tst.l d0      * versch. Vorzeichen
  rts
fcmp2:          * gleiche Vorzeichen
tst.l d0
bmi.s fcmp4
  cmp d3,d0     * positive Zahlen
  bne.s fcmp9
  cmp.l d4,d1
  rts
fcmp4:
  cmp d0,d3     * negative Zahlen
  bne.s fcmp9
  cmp.l d1,d4
fcmp9:
rts


*******************************************
* FSQRT   Wurzelziehen                    *
* Argument und Ergebnis in (d0,d1)        *
* Zerstörte Register:   d3,d4,d6,d7       *
*******************************************
sqrtit equ 3
*
fsqrt:
tst.l d1
beq fzero
tst.l d0
bmi fnan
link a0,#-16
  radikand equ -16
  altwurz  equ -8
move.l d0,radikand(a0)
move.l d1,radikand+4(a0)
asr.l #1,d1
asr #1,d0
bcs.s sqrt1
  bclr #30,d1
sqrt1:          * erste Iteration kann mit INTEGER erfolgen!!!
move #sqrtit-1,d6
sqrtloop:
  move d6,-(a7)
  move.l d0,altwurz(a0)
  move.l d1,altwurz+4(a0)
  move.l d0,d3
  move.l d1,d4
  move.l radikand(a0),d0
  move.l radikand+4(a0),d1
  bsr fdiv
  move.l altwurz(a0),d3
  move.l altwurz+4(a0),d4
  bsr fadd
  subq #1,d0
  move (a7)+,d6
  dbra d6,sqrtloop
unlk a0
rts


*******************************************
* FEXP    Exponentialfunktion             *
* Argument und Ergebnis in (d0,d1)        *
*                                         *
*******************************************
fexp:
link a0,#-16
  xexp   equ -16
  xganz  equ -8
```

```
clr.l d3        * 1/ln(2) = log2(e)
move.l #$b8aa3b29,d4
bsr fmul
move.l d0,xganz(a0)
move.l d1,xganz+4(a0)
bsr ffrac               * aus [0..ln(2)]
lea poly_exp(pc),a1
bsr horner
move.l d0,xexp(a0)
move.l d1,xexp+4(a0)
move.l xganz(a0),d0
move.l xganz+4(a0),d1
bsr ftrunc
move.l d1,d4
move.l xexp(a0),d0
move.l xexp+4(a0),d1
add d4,d0               * skalieren
exp99:
unlk a0
rts

macro makereal
dc.w (|0&$8000),(|0&$7ff0)/$10-$3ff
dc.w $8000+(|0&$f)*$800+(|1&$ffe0)/$20
dc.w (|1&$1f)*$800+(|2+$0010)/$20
endmacro

poly_exp:
.MAKEREAL $3EBF,$7876,$799D,$7880  *  1.87579142e-006
.MAKEREAL $3EED,$B566,$B8FA,$6FA7  *  1.41661638e-005
.MAKEREAL $3F24,$5522,$72C8,$43E4  *  1.55125098e-004
.MAKEREAL $3F55,$D5F3,$D46A,$9CEA  *  1.33274852e-003
.MAKEREAL $3F83,$B2C4,$5BD5,$D683  *  9.61831479e-003
.MAKEREAL $3FAC,$6B07,$E5A3,$5025  *  5.55040806e-002
.MAKEREAL $3FCE,$BFBE,$02D5,$3ED0  *  2.40226509e-001
.MAKEREAL $3FE6,$2E42,$FEFA,$39EF  *  6.93147181e-001
.MAKEREAL $3FF0,$0000,$0000,$0000  *  1.00000000e+000
dc.w 1

horner:
move.l d1,-(a7)     * x auf den Stack
move.l d0,-(a7)
move.l (a1)+,d0
move.l (a1)+,d1
horn2:
  cmp #1,(a1)       * Endekennzeichen
  beq.s horn9
  move.l (a7),d3
  move.l 4(a7),d4
  bsr fmul
  move.l (a1)+,d3
  move.l (a1)+,d4
  bsr fadd
  bra.s horn2
horn9:
addq #8,a7      * x vom Stack nehmen
rts


*************************************************************************
*****

wrhex:
.save
lea fpuffer(pc),a0
move #!print8x,d7
trap #1
lea fpuffer(pc),a0
move #7,d7
trap #6  * JADOS schreibe
.rts0

fpuffer:
```

```
ds.b 50


.endcode
.endmodul

*********************************************************************

END
BEGIN


MACRO ffunc
  _f|0:
  f|1.d rsaved(a7),fp0
  fmove.d fp0,rsaved(a7)
  rts
ENDMACRO

.modul .Transzendent
.segmentc CODE,CODE-GROUP

* .globproc _FEXP
.globproc _FLN
.globproc _FSIN
.globproc _FCOS
.globproc _FTAN
.globproc _FARCSIN
.globproc _FARCCOS
.globproc _FARCTAN
.globproc _FSINH
.globproc _FCOSH
.globproc _FTANH

.code

* .ffunc exp,etox
.ffunc sin,sin
.ffunc cos,cos
.ffunc tan,tan
.ffunc sinh,sinh
.ffunc cosh,cosh
.ffunc tanh,tanh
.ffunc arcsin,asin
.ffunc arccos,acos
.ffunc arctan,atan
.ffunc ln,logn

.endcode
.endmodul

*************************************************************************
*****

END
BEGIN


.modul .GETKEY
.segmentc CODE,CODE-GROUP

.globproc _KEYPRESS
.globproc _GETKEY

.code

_keypress:
.save
.grundq csts
and #1,d0
move.b d0,saved(a7)     * Funktionswert
.rts0
```

```
_getkey:
.save
.grundq ci
move.b d0,saved(a7)     * Funktionswert
.rts0

.endcode
.endmodul

*******************************************************************************
*****

END
BEGIN


.modul .Strings
.segmentc CODE,CODE-GROUP

.globproc STRUPPER
.globproc STRAPPEND
.globproc STRPOS
.globproc SUBSTRING
.globproc STRDELETE
* .globproc STRINSERT

.code

strupper:
.save
movea.l saved(a7),a0
clr d0
move.b (a0)+,d0
subq #1,d0
bmi.s str9upp
str2upp:
  cmp.b #'a',(a0)
  blo.s str3upp
  cmp.b #'z',(a0)
  bhi.s str3upp
    add.b #'A'-'a',(a0)
  str3upp:
  addq.l #1,a0
  dbra d0,str2upp
str9upp:
.rtsq 4

_anz  equ saved
_pos  equ saved+2
_sd1  equ saved+4

strdelete:
.save
movea.l _sd1(a7),a0
clr d0
move.b (a0),d0
move _pos(a7),d1
bgt.s str2del
  moveq #1,d1           * pos >= 1
str2del:
cmp d0,d1
bgt.s str99del          * pos > Länge => fertig
move _anz(a7),d2
ble.s str99del          * anz <= 0 => fertig
movea.l a0,a2
adda d1,a0              * Zielposition
movea.l a0,a1
adda d2,a1              * Quellposition
sub d2,d0
sub d1,d0               * Restlänge
bpl.s str5del
  subq #1,d1
  move.b d1,(a2)
```

```
  bra.s str99del
str5del:
  sub.b d2,(a2)
  str6del:
    move.b (a1)+,(a0)+
    dbra d0,str6del
str99del:
.rtsq 8


_ss2  equ saved
_anzahl  equu saved+4
_von  equ saved+6
_ss1  equ saved+8

substring:
.save
lea _ss1(a7),a0          * Quelle
clr d0
move.b (a0),d0           * Länge Quelle
movea.l _ss2(a7),a1      * Ziel
clr.b (a1)               * Länge Ziel := 0
move _von(a7),d2
bgt.s sub2s
  moveq #1,d2            * von >= 1
sub2s:
cmp d0,d2
bgt.s sub99s             * von > Länge ==> fertig
move _anzahl(a7),d3
blt.s sub99s
add d2,d3
subq #1,d3               * bis
cmp d0,d3
ble.s sub3s
  move d0,d3             * bis <= Länge
sub3s:
adda d2,a0               * Startposition
sub d2,d3                * Anzahl-1
move.b d3,(a1)
addq.b #1,(a1)+          * Anzahl-1+1
sub4s:
  move.b (a0)+,(a1)+
  dbra d3,sub4s
sub99s:
.rts 90


strappend:
.save
lea saved+4(a7),a0
clr d0
move.b (a0)+,d0
movea.l saved(a7),a1
clr d1
move.b (a1),d1
add d0,d1
sub #81,d1
ble.s str3app
  sub d1,d0
str3app:
clr d1
move.b (a1),d1
add.b d0,(a1)
subq #1,d0
bmi.s str9app
adda d1,a1
addq.l #1,a1
str5app:
  move.b (a0)+,(a1)+
  dbra d0,str5app
str9app:
.rts 82+4
```

```
strpos:
.save
lea saved(a7),a0        * hier wird gesucht (Quellstring)
clr d1
move.b (a0),d1          * Länge Quellstring
clr d2
move.b saved+82(a7),d2  * Länge Suchstring
bne.s str1pos
  moveq #1,d4
  bra.s str9pos
str1pos:
subq #1,d2
sub d2,d1               * hinterster Suchanfang
ble.s str99pos
clr d4                  * vorne anfangen
str2pos:
  addq #1,d4
  addq #1,a0
  movea.l a0,a2
  lea saved+83(a7),a1   * dies wird gesucht
  move d2,d3
  str3pos:
    move.b (a1)+,d0
    cmp.b (a2)+,d0
    dbne d3,str3pos
  beq.s str9pos
  cmp d1,d4
  blt.s str2pos
str99pos:
clr d4
str9pos:
move d4,saved+2*82(a7)
.rts 2*82


.endcode
.endmodul

************************************************************************
****

END
BEGIN


.modul .GRUNDPROGRAMM
.segmentc CODE,CODE-GROUP

.globproc GRUND
.globproc GRUND1
.globproc JADOS
.globproc G_VERSION
.globproc CLRSCREEN
.globproc GETLINE
.globproc CLREOLN
.globproc RANDOM
.globproc SIZE

.code

_pointer equ saved

grund1:
.save
move _int1(a7),d0
move _int2(a7),d1
move _int0(a7),d7
trap #1
.rtsq 6

grund:
.save
```

```
movea.l _pointer(a7),a6
movem.w (a6)+,d0-d7
movem.l (a6)+,a0-a5
movea.l (a6),a6
move _int0(a7),d7
trap #1
movea.l _pointer(a7),a6
adda #8*2+6*4,a6
movem.l a0-a5,-(a6)
movem.w d0-d7,-(a6)
.rtsq 6

jados:
.save
movea.l _pointer(a7),a6
movem.w (a6)+,d0-d7
movem.l (a6)+,a0-a5
movea.l (a6),a6
move _int0(a7),d7
trap #6
movea.l _pointer(a7),a6
adda #8*2+6*4,a6
movem.l a0-a5,-(a6)
movem.w d0-d7,-(a6)
.rtsq 6

g_version:
.save
.grundq getvers
movea.l saved(a7),a0
move.b #4,(a0)+
move d0,d1
lsr #8,d1
and #$f,d1
add #'0',d1
move.b d1,(a0)+
move.b #'.',(a0)+
move d0,d1
lsr #4,d1
and #$f,d1
add #'0',d1
move.b d1,(a0)+
and #$0f,d0
add #'0',d0
move.b d0,(a0)+
.rtsq 4

clrscreen:
.save
.grundq clrscreen
.rts0

getline:
.save
move saved+4(a7),d0     * Zeilennummer
.grundq getline
clr d0
move.b (a1),d0          * Anzahl Zeichen
movea.l saved(a7),a3    * hier wird hinkopiert
move.b d0,(a3)+
subq #1,d0
bmi.s get9line
get2line:
  move.b (a0)+,(a3)+
  dbra d0,get2line
get9line:
.rtsq 6

clreoln:
.save
moveq #$1b,d0
.grundq co2
moveq #'T',d0
```

```
.grundq co2
.rts0

random:
.save
move _int2(a7),d0
.grundq rnd
move d0,_int1(a7)
.rtsq 2

size:
.save
move saved(a7),d0
.grundq size
.rtsq 2


.endcode
.endmodul


*****************************************************************************
****

END
BEGIN


.modul .Grund-Cursor
.segmentc CODE,CODE-GROUP

.globproc CURON
.globproc CUROFF
.globproc CURSEIN
.globproc CURSAUS
.globproc SETCURXY
.globproc GETCURXY

.code

curon:
.save
.grundq curon
.rts0

curoff:
.save
.grundq curoff
.rts0

cursein:
.save
.grundq cursein
.rts0

cursaus:
.save
.grundq cursaus
.rts0

setcurxy:
.save
move _int1(a7),d1
move _int2(a7),d2
.grundq setcurxy
.rtsq 4

getcurxy:
.save
.grundq getcurxy
movea.l _var1(a7),a0
move d1,(a0)
movea.l _var2(a7),a0
```

```
move d2,(a0)
.rtsq 8

.endcode
.endmodul


************************************************************

END
BEGIN


.modul .Symboltabelle
.segmentc CODE,CODE-GROUP

.globproc SYMCLR
.globproc ZUWEIS
.globproc WERT
.globproc FPUWERT

.extproc _convstr

.code

symclr:
.save
.grundq symclr
.rts0

zuweis:
.save
lea saved(a7),a0
bsr _convstr
.grundq zuweis
.rts 82

wert:
.save
lea saved(a7),a0
bsr _convstr
.grundq wert
move d0,saved+82(a7)
.rts 82

fpuwert:        * geht nur mit 68020/68881
.save
lea saved(a7),a0
bsr _convstr
.grund fpuwert
fmove.d fp0,saved+82(a7)
.rts 82

.endcode
.endmodul

************************************************************

END
BEGIN


.modul .Synchronisation
.segmentc CODE,CODE-GROUP

.globproc DELAY
.globproc SYNC

.code

delay:
.save
clr.l d0
```

```
move saved(a7),d0
.grundq delay
.rtsq 2

sync:
.save
.grundq sync
and #1,d0
move.b d0,saved(a7)
.rts0

.endcode
.endmodul

********************** Grundprogramm-Grafik ******************

END
BEGIN


.modul .GRUND-GRAFIK
.segmentc CODE,CODE-GROUP

.globproc MOVETO
.globproc DRAWTO
.globproc ERAPEN
.globproc SETPEN
.globproc LINE,          *
.globproc BAR,           *
.globproc RECTANGLE,     *
.globproc CIRCLE,        *
.globproc SETXOR
.globproc NEWPAGE
.globproc SETFLIP
.globproc AUTOFLIP
.globproc CLPG
.globproc CLR
.globproc PROGZGE

.code

moveto:
.save
move _int1(a7),d1
move _int2(a7),d2
asr #1,d2
.grundq moveto
.rtsq 4

drawto:
.save
move _int1(a7),d1
move _int2(a7),d2
asr #1,d2
.grundq drawto
.rtsq 4

arc:

bar:
.save
move _ra1(a7),d5
move _ra2(a7),d6
move _ra3(a7),d3
move _ra4(a7),d4
asr #1,d6
asr #1,d4
cmp d4,d6
bgt.s bar2
  exg d4,d6     * jetzt ist d6>=d4
bar2:
move d4,d2
bar3:
```

```
  move d5,d1
  .grundq moveto
  move d3,d1
  .grundq drawto
  addq #1,d2
  cmp d6,d2
  blt.s bar3
.rts 8

_cx     equ saved+4
_cy     equ saved+2
_cr     equ saved

circle:
.save
move _cx(a7),d3
move _cy(a7),d4
asr #1,d4
move _cr(a7),d5
bpl.s _circle2
  neg d5
_circle2:
move d3,d1
add d5,d1
move d4,d2
.grundq moveto
move #360,-(a7)
moveq #30,d6
cmp #4,d5
ble _circle3
moveq #15,d6
cmp #10,d5
ble _circle3
moveq #10,d6
cmp #30,d5
ble _circle3
moveq #5,d6
cmp #60,d5
ble _circle3
moveq #5,d6

_circle3:
  move (a7),d0
  sub d6,d0
  move d0,(a7)
  .grundq sin
  muls d5,d0
  asr.l #8,d0
  asr.l #1,d0
  move d0,d2
  add d4,d2
  move (a7),d0
  .grundq cos
  muls d5,d0
  asr.l #8,d0
  move d0,d1
  add d3,d1
  .grundq drawto
  tst (a7)
  bne _circle3
move (a7)+,d0
.rts 6

_ra1    equ saved+6
_ra2    equ saved+4
_ra3    equ saved+2
_ra4    equ saved

rectangle:
.save
move _ra1(a7),d5
move _ra2(a7),d6
move _ra3(a7),d3
```

```
move _ra4(a7),d4
asr #1,d6
asr #1,d4
move d5,d1
move d6,d2
.grundq moveto
move d3,d1
.grundq drawto
move d4,d2
.grundq drawto
move d5,d1
.grundq drawto
move d6,d2
.grundq drawto
.rts 8

line:
.save
move _ra1(a7),d1
move _ra2(a7),d2
asr #1,d2
.grundq moveto
move _ra3(a7),d1
move _ra4(a7),d2
asr #1,d2
.grundq drawto
.rts 8

setpen:
.save
.grundq setpen
.rts0

erapen:
.save
.grundq erapen
.rts0

setxor:
.save
move.b saved(a7),d0
.grundq setxor
.rtsq 2

newpage:
.save
move _int1(a7),d0
move _int2(a7),d1
.grundq newpage
.rtsq 4

setflip:
.save
move _int1(a7),d0
move _int2(a7),d1
.grundq setflip
.rtsq 4

autoflip:
.save
.grundq autoflip
.rts0

clpg:
.save
.grundq clpg
.rts0

clr:
.save
.grundq clr
.rts0
```

```
progzge:
.save
movea.l saved(a7),a1
lea zgebuff(pc),a0
move.b 1(a1),0(a0)      * von Wort auf Byte umspeichern
move.b 3(a1),1(a0)
move.b 5(a1),2(a0)
move.b 7(a1),3(a0)
move.b 9(a1),4(a0)
.grundq progzge
.rtsq 4

zgebuff:
ds.b 6


.endcode
.endmodul

************************************************************

END
BEGIN


.modul .Text-Ein/Aus
.segmentc CODE,CODE-GROUP

.globproc G_READ
.globproc READAUS
.globproc G_WRITE

.extproc _convstr

.code

_g_w1   equ saved+82+2+2
_g_w2   equ saved+82+2
_g_w3   equ saved+82
_g_w4   equ saved

g_write:
.save
lea _g_w4(a7),a0
bsr _convstr
move _g_w1(a7),d0
move _g_w2(a7),d1
move _g_w3(a7),d2
asr #1,d2
.grundq write
.rts 82+2+2+2

_g_r1   equ saved+10
_g_r2   equ saved+8
_g_r3   equ saved+6
_g_r4   equ saved+4
_g_r5   equ saved

g_read:
.save
move _g_r1(a7),d0
move _g_r2(a7),d1
move _g_r3(a7),d2
asr #1,d2
move _g_r4(a7),d3
movea.l _g_r5(a7),a1
lea 1(a1),a0
.grundq read
move.b d4,(a1)  * Stringlänge noch eintragen
.rts 12

readaus:
.save
```

```
move _g_r1(a7),d0
move _g_r2(a7),d1
move _g_r3(a7),d2
asr #1,d2
move _g_r4(a7),d3
movea.l _g_r5(a7),a1
lea 1(a1),a0
.grundq readaus
move.b d4,(a1)  * Stringlänge noch eintragen
.rts 12

.endcode
.endmodul

************************************************************

END
BEGIN


.modul .Uhr-Funktionen
.segmentc CODE,CODE-GROUP

.globproc UHRPRINT
.globproc GETTIME
.globproc GETDATE

.code

_up1    equ saved+4
_up2    equ saved

uhrprint:
.save
move _up1(a7),d0
movea.l _up2(a7),a1
lea 1(a1),a0
.grund uhrprint
move.l a0,d0
sub.l a1,d0
subq #1,d0
move.b d0,(a1)
.rts 6


_tim1   equ saved+4*3
_tim2   equ saved+4*2
_tim3   equ saved+4
_tim4   equ saved

bcdtoint:
move d0,d1
and #%1111,d1
lsr #4,d0
and #%1111,d0
mulu #10,d0
add d1,d0
rts

macro bcdtime
  move.b |0(a0),d0
  bsr bcdtoint
  movea.l |1(a7),a1
  move d0,(a1)
endmacro

gettime:
.save
lea uhrpuffer(pc),a0
.jados getuhr
.bcdtime 0,_tim1
.bcdtime 1,_tim2
.bcdtime 6,_tim3
```

```
.bcdtime 7,_tim4
.rts 4*4

getdate:
.save
lea uhrpuffer(pc),a0
.jados getuhr
.bcdtime 4,_tim1
.bcdtime 3,_tim2
.bcdtime 2,_tim3
.bcdtime 5,_tim4
.rts 4*4

uhrpuffer: ds.b 10

.endcode
.endmodul

************************************************************

END
BEGIN


.modul .Baugruppen
.segmentc CODE,CODE-GROUP

.globproc REL
.globproc GETAD8
.globproc SETDA
.globproc GDPVERS
.globproc SERINIT

.code

rel:
.save
tst.b saved(a7)
beq.s _relaus
  .grundq relan
  bra.s _rel9
_relaus:
  .grundq relaus
_rel9:
.rtsq 2

getad8:
.save
move saved(a7),d0
.grundq getad8
move d0,saved+2(a7)
.rtsq 2

setda:
.save
move saved+2(a7),d1
move saved(a7),d2
.grundq setda
.rtsq 4

gdpvers:
.save
.grundq gdpvers
move d0,saved(a7)
.rts0

serinit:
.save
move saved+2(a7),d0
move saved(a7),d1
.grundq siinit
.rtsq 4
```

```
.endcode
.endmodul

************************************************************

END
BEGIN


.modul .Schildkroete
.segmentc CODE,CODE-GROUP

.globproc SCHREITE
.globproc SCHR16TEL
.globproc DREHE
.globproc HEBE
.globproc SENKE
.globproc GRAPSET
.globproc FIRSTTIME
.globproc GRAPOFF
.globproc GRAPHIDE
.globproc GRAPSHOW
.globproc AUFXY
.globproc AUFK

.code

schreite:
.save
move saved(a7),d0
.grundq schreite
.rtsq 2

schr16tel:
.save
move saved(a7),d0
.grundq schr16tel
.rtsq 2

drehe:
.save
move saved(a7),d0
.grundq drehe
.rtsq 2

hebe:
.save
.grundq hebe
.rts0

senke:
.save
.grundq senke
.rts0

grapset:
.save
move saved(a7),d3
move saved+2(a7),d2
move saved+4(a7),d1
.grundq set
.rtsq 6

firsttime:
.save
.grundq firsttime
.rts0

grapoff:
.save
.grund grapoff
.rts0
```

```
graphide:
.save
.grundq hide
.rts0

grapshow:
.save
.grundq show
.rts0

aufxy:
.save
move saved(a7),d2
move saved+2(a7),d1
.grundq aufxy
.rtsq 4

aufk:
.save
move saved(a7),d0
.grundq aufk
.rtsq 2


.endcode
.endmodul

************************************************************

END
BEGIN


.modul .JADOS-Funktionen
.segmentc CODE,CODE-GROUP

.globproc SOUND
.globproc BEEP
.globproc BELL
.globproc ERRNOISE
.globproc WRITECMD
.globproc RESPONSE
.globproc CPU
.globproc MOTOROFF
.globproc INPORT
.globproc OUTPORT
* .globproc LINEEDIT
.globproc DISKINFO
.globproc FILEINFO
.globproc GETDPATH
.globproc GETHDISK
.globproc J_VERSION

.globproc DRIVECODE
.globproc FLOPPYRD
.globproc FLOPPYWR
.globproc GETDRIVE
.globproc SETDRIVE

.extproc _convstr

.code

writecmd:
.save
lea saved(a7),a0
bsr _convstr
.jados wrtcmd
.rts 82

cpu:
.save
.jados getcpu
```

```
move d0,saved(a7)
.rts0

motoroff:
.save
.jados motoff
.rts0

bell:
.save
.jados jbell
.rts0

errnoise:
.save
.jados jerrnoise
.rts0

response:
.save
.jados jresponse
.rts0

sound:
.save
lea pufferj(pc),a0
movea.l saved(a7),a1
moveq #15,d1
_sound2:
  move.w (a1)+,d0
  move.b d0,(a0)+
  dbra d1,_sound2
lea pufferj(pc),a0
.jados jsound
.rtsq 4

beep:
.save
move _int1(a7),d1
ext.l d1
move _int2(a7),d2
ext.l d2
.jados jbeep
.rtsq 4

inport:
.save
move.w saved+4(a7),d1      * Portadresse
or.l #$ffffff00,d1
clr d0
.jados jinport
movea.l saved(a7),a1
move.w d0,(a1)
.rtsq 4+2

outport:
.save
move.w _int1(a7),d1
or.l #$ffffff00,d1
move.w _int2(a7),d0
.jados joutport
.rtsq 2+2


getdpath:
.save
.jados jgetdpath
movea.l saved(a7),a1
lea 1(a1),a2
clr d1
get2dp:
  move.b (a0)+,d0
  cmp.b #$ff,d0         * Ende ?
```

```
  beq.s get8dp
  addq #1,d1
  .jados numtoasc
  move.b d0,(a2)+
  bra.s get2dp
get8dp:
move.b d1,(a1)
.rtsq 4

gethdisk:
.save
.jados jgethdisk
move #2,d2
clr d0
move.b saved(a7),d0
cmp #'Z',d0
bgt.s get9hd
sub #'A',d0
bmi.s get9hd
  asl #3,d0   * mal 8
  move 0(a0,d0),d2
get9hd:
move d2,saved+2(a7)
.rtsq 2

diskinfo:
.save
move.b saved+4(a7),d0   * Laufwerk
.jados asctonum
move.b d0,d4
lea pufferj(pc),a0
.jados jdiskinfo
not d0
and #1,d0
move.b d0,saved+6(a7)   * Fehlercode (Funktionswert)
beq.s disk9info
movea.l saved(a7),a1    * Adresse VAR-Parm
move #5,d1
disk2info:
  move.l (a0)+,d0
  move.w d0,(a1)+
  dbra d1,disk2info
move.l (a0)+,d0
divu #1024,d0
move.w d0,(a1)+
disk9info:
.rtsq 6


fileinfo:
.save
lea saved+4(a7),a0
bsr _convstr
lea pufferj(pc),a1
.jados fillfcb
tst.b d0
bne file9info
.jados jfileinfo
tst.b d0
bne file9info
  movea.l saved(a7),a1
  divu #1024,d1
  move d1,(a1)+

  move.b d2,d1
  bsr bcd2a1p
  move.b d2,d1
  bsr bcd2a1p
  move.b d2,d1
  bsr bcd2a1p

  ext.w d3
  move d3,(a1)+
```

```
file9info:
not d0
and #1,d0
move.b d0,saved+86(a7)   * Fehlercode
.rts 82+4


bcd2a1p:        * BCD in d1 als Integer nach (a1)+, d2 >> 8
move d1,d7
and #%1111,d7
lsr #1,d1
and #%01111000,d1       * 8 mal Zifferwert
add d1,d7
lsr #2,d1               * 2 mal Zifferwert
add d1,d7
move d7,(a1)+
lsr.l #8,d2
rts



j_version:
.save
.jados getversi
movea.l saved(a7),a0
move.b #4,(a0)+
rol.l #8,d0
move.b d0,(a0)+
rol.l #8,d0
move.b d0,(a0)+
rol.l #8,d0
move.b d0,(a0)+
rol.l #8,d0
move.b d0,(a0)+
.rtsq 4

drivecode:
.save
move.b saved(a7),d0
.jados asctonum
clr d4
move.b d0,d4
.jados jdrivecod
move d4,saved+2(a7)
.rtsq 2

getdrive:
.save
.jados jgetdrive
.jados numtoasc
move.b d0,saved(a7)
.rts0

setdrive:
.save
move.b saved(a7),d0
.jados asctonum
and #$ff,d0
.jados jsetdrive
.rtsq 2

floppyrd:
.save
moveq #1,d0             * Sektor lesen
bra.s floppyx

floppywr:
.save
moveq #2,d0             * Sektor schreiben
floppyx:
movea.l saved(a7),a0    * Ladeadresse
move saved+4(a7),d2     * Sektor
move saved+6(a7),d3     * Spur
```

```
move saved+8(a7),d4     * Laufwerkscode
.jados floppy
not d0
and #1,d0
move.b d0,saved+10(a7)  * Fehlercode
.rts 10


pufferj: ds.b 48

.endcode
.endmodul


************************************************************


END
BEGIN

.modul .JADOS-Retcode
.segmentc CODE,CODE-GROUP

.globproc GETPARM
.globproc SETRETCODE
.globproc GETRETCODE

.code


getretcode:
.save
.jados gtretcod
movea.l saved(a7),a0
move d0,(a0)
.rtsq 4

setretcode:
.save
move saved(a7),d0
ext.l d0
.jados stretcod
.rtsq 2

_gp1    equ saved+4     * Nr.
_gp2    equ saved       * String

getparm:
.save
move.w _gp1(a7),d0
movea.l _gp2(a7),a1
lea 1(a1),a2
moveq #getparm0,d7
trap #6
tst d0
ble _getp9
get2parm:               * Leerzeichen ueberspringen
  cmp.b #' ',(a0)
  bne.s get3parm
  addq #1,a0
  bra.s get2parm
get3parm:
subq #1,d0
beq get6parm
get4parm:               * Parameter ueberspringen
  cmp.b #' ',(a0)
  beq.s get2parm
  tst.b (a0)
  beq _getp9
  addq #1,a0
  bra.s get4parm
get6parm:               * nur einen Parameter kopieren
moveq #-1,d0
```

```
get7parm:
  addq #1,d0
  move.b (a0)+,(a2)+
  beq get81parm
  cmp.b #' ',-1(a2)
  bne.s get7parm
bra.s get8parm

_getp9:                 * gesamten Parameterstring kopieren
moveq #-1,d0
_getp3:
  addq #1,d0
  move.b (a0)+,(a2)+
  bne.s _getp3
get8parm:
clr.b -1(a2)
get81parm:
move.b d0,(a1)
.rtsq 2+4


.endcode
.endmodul


.endlib


END
```