```
     d
     d
     d
 ddd d   y    y    eeee    r rrr
d    dd   y    y   e    e   rr    r
d    d   y    y   eeeeee   r
d    d   y    y   e        r
d    dd   y   yy   e    e   r
 ddd d    yyy y    eeee    r
              y
         y    y
          yyyy
```

```
                    ll
                     l
                     l
r rrr     eeee       l      oooo     cccc              ssss
rr   r   e    e      l     o    o   c    c            s    s
r        eeeeee      l     o    o   c                  ss
r        e           l     o    o   c                    ss
r        e    e      l     o    o   c    c     ..     s    s
r         eeee      lll     oooo     cccc      ..      ssss
```

(BIOS LISTING).

```
                     i

u    u   n nnn      ii     x    x
u    u   nn   n      i      x  x
u    u   n    n      i       xx
u    u   n    n      i       xx
u   uu   n    n      i      x  x
 uuu u   n    n     iii    x    x
```

```
Job:   reloc.s
Date:  Tue Jul  9 12:43:17 1985
```

```
*---- Conditional assembly equates:
rom             =       0               ; generate ROMable system
ram             =       1               ; generate loadable system

usa             =       0               ; for USA
germany         =       1               ; for Deutchland
france          =       2               ; for France
uk              =       3               ; for Britain


*---- Version information:
version         equ     $0100           ; system version number (VVRR)
date            equ     $06141985       ; date system was built


*---- Conditional assembly switches:
systype         =       ram             ; type of system
country         =       usa             ; country



*+
*  Parameters for RAM and ROM systems;
*  Adjust these equates with system size and location changes.
*  'endos' points to the last bit of RAM the system uses (plus one).
*  'the_magic' points to a parameter block containing information
*  about the location of the AES, and how much RAM it uses.
*
*-
 ifeq systype-rom                       * For ROM:
endos           equ     $5000           ; end of OS memory usage
the_magic       equ     $fefff4         ; -> magic stuff (top of the ROM)
 endc

 ifeq systype-ram                       * For RAM:
endos           equ     $19c00          ; end of OS memory usage
the_magic       equ     endos-$c        ; -> magic stuff (at the top of the OS)
 endc



*-----------------------------------------------------------------------
*                                                                       :
*       ST Series BIOS                                                  :
*       (C)1985 Atari Corp.                                             :
*       All Rights Reserved.                                            :
*                                                                       :
*       System Initialization                                           :
*       ROM header                                                      :
*       RAM variable equates                                            :
*                                                                       :
*                                                                       :
*---- Edit history:                                                     :
*                                                                       :
*  (lost history)       [From Oct '84, incarnations as part of          :
*                        the debugger cart, and the CP/M-68K BIOS]      :
```

```
* 02-Feb-1985 lmd        Converted from CP/M BIOS.                      :
* 24-Feb-1985 lmd        I munge this file every day.                   :
* 25-Feb-1985 lmd        Added _cmdload flag (load COMMAND.COM from disk):
* 25-Feb-1985 lmd        Changed _get_mpb, added "hard_reset" conditional:
*                        assembly switch.                               :
* 27-Feb-1985 lmd        Added hard disk hooks.                         :
*  1-Mar-1985 lmd        Added _supstk (from GEMDOS)                    :
*  1-Mar-1985 lmd        Added _mediach BIOS function                   :
*  4-Mar-1985 lmd        Added 'cartscan' and associated calls to it    :
*  7-Mar-1985 lmd        Integrate new character I/O                    :
*  8-Mar-1985 lmd        Critical error handler, random trap hacking    :
*  9-Mar-1985 lmd        BIOS traps are re-entrant to 3 levels, and     :
*                        callable from user mode.                       :
* 10-Mar-1985 lmd        Consolidated BSS, installed "extended" traps   :
* 15-Mar-1985 lmd        Re-integration with RBIOS.  200hz raw sysTick  :
* 16-Mar-1985 lmd        Warmstart banished.  Procdump on uncaught traps :
* 27-Mar-1985 lmd        Added "_scrdmp" trap#14 function               :
* 27-Mar-1985 lmd        Added "getshift" trap#13 function              :
*  8-Apr-1985 lmd        Re-integration with serial code                :
*  8-Apr-1985 lmd        Moved floppy/FIFO lock to public basepage      :
*  9-Apr-1985 lmd        get/set shift bits (trap #13, $0b)             :
*  9-Apr-1985 lmd        Added _dskbufp -> _diskbuf                     :
* 13-Apr-1985 lmd        Added _autopath (autoexec path pointer)        :
* 15-Apr-1985 lmd        Happy IRS day.                                 :
* 15-Apr-1985 lmd        Moved _vblqueue to low memory (thank Ghu!)     :
* 17-Apr-1985 lmd        Added _prtblk primitive                        :
* 17-Apr-1985 lmd        Hblank (vector interrupt #2) hacks caller's IPL :
*                        to 3.                                          :
*  1-May-1985 lmd        Added supexec() & wvbl() extended functions    :
*  8-May-1985 lmd        RAM-loaded system wired 'memcntrl' to 512K;    :
*                        now it takes whatever the boot ROMs give it.   :
*  9-May-1985 lmd        Added _asc_out to character device table.      :
* 15-May-1985 lmd        Moved _cursconf to escape module.              :
* 23-May-1985 lmd        Added 'magic' parameter -- makes it easy to    :
*                        blow the GEM AES away.  Huzzah!                :
* 24-May-1985 lmd        Added mushroom cloud display on processor      :
*                        exception, out of sheer boredom.               :
* 28-May-1985 lmd        Added new _prtblk.  Screen dump understands high:
*                        quality print mode.                            :
*                                                                       :
*-----------------------------------------------------------------------
                 text




*--------------- Exports:
        .globl  endosbss                ; (informative) end OS bss
        .globl  _dumpflg,_prtcnt        ; screen dump flag (& its alias)
        .globl  _prtabt                 ; printer abort flag
        .globl  flock                   ; floppy/FIFO lock
        .globl  sshiftmd                ; shiftmd shadow
        .globl  etv_timer               ; timer handoff vector
        .globl  _membot                 ; (best guess) bottom of TPA
        .globl  _memtop                 ; top of TPA (first unusable byte)
        .globl  _timr_ms                ; system timer calibration (in ms)
        .globl  _vblqueue               ; vbl queue
        .globl  _vbclock                ; count of unblocked vblank interrupts
        .globl  _frclock                ; count of all vblank interrupts
        .globl  _v_bas_ad               ; video base addr
        .globl  con_state               ; state of conout() parser
        .globl  save_row                ; saved row# for cursor X-Y addressing
        .globl  _bufl                   ; two buffer-list headers
        .globl  _bootdev                ; default boot device [0]
        .globl  _cmdload                ; nonzero: exec shell on boot device
        .globl  conterm                 ; terminal emulator bitSwitches
        .globl  _nflops                 ; "Hey!  Clams got floppies!"
        .globl  _critic                 ; critical error handler binding for C
        .globl  _hz_200                 ; 200hz raw system timer tick
        .globl  seekrate                ; default floppy seek rate
        .globl _fverify                 ; nonzero: verify on floppy write
        .globl _drvbits                 ; long bitmap of block devices
        .globl  conterm                 ; console/vt52 bits

        .globl  _hinit                  ; go through hdv_init
        .globl _dskboot                 ; boot from somewhere
        .globl  _fastcpy                ; fast copy (for unaligned DMA)



*-------------- Imports:
        .globl  _cursconf               ; cursor configuration
        .globl  _asc_out                ; "raw" character output to screen
        .globl  pconfig                 ; printer configuration word
        .globl  _prtblk                 ; _prtblk primitive
        .globl  esce                    ; {escape.s} "hard" turn on cursor
        .globl  _osi                    ; initialize OS
        .globl  initmfp                 ; init character I/O
        .globl  esc_init                ; init glass tty
        .globl  initmous                ; mouse vector init
        .globl  _mediach                ; media change inquiry
        .globl  _proto_bt               ; prototype boot sector
        .globl  _flopwr                 ; write sector(s)
        .globl  _flopver                ; verify sector(s)
        .globl  _flopfmt                ; format track
        .globl  _rand                   ; generate random number

        .globl auxistat                 ; input-status
        .globl constat
        .globl midstat
        .globl _lstin                   ; input
        .globl auxin
        .globl conin
        .globl midin
        .globl _lstostat                ; output-status
        .globl _auxostat
        .globl conoutst
        .globl ikbdost
        .globl midiost
        .globl _lstout                  ; output
        .globl _auxout
        .globl conout
        .globl midiwc
        .globl ikbdwc

        .globl midiws                   ; write MIDI string
        .globl mfpint                   ; setup MFP interrupt
        .globl iorec                    ; configure I/O record
        .globl rsconf                   ; configure RS-232
        .globl keytrans                 ; store keyboard translation
        .globl settime                  ; set ikbd date
        .globl gettime                  ; get ikbd date
        .globl bioskeys                 ; reset keyboard to power-up defaults
        .globl ikbdws                   ; write string to ikbd

        .globl  line1010                ; line 1010 handler
        .globl  kbshift                 ; keyboard shift status

        .globl  jdisint
        .globl  jenabint
        .globl  giaccess
        .globl  offgibit
        .globl  ongibit
        .globl  xbtimer
        .globl  dosound
        .globl  setprt
        .globl  kbrate
        .globl  ikbdvecs


        .globl  _supstk                 ; GEMDOS super stack
        .globl  _diskbuf                ; disk buffer

        .globl  _getdsb                 ; return disk's state pointer
        .globl  _boot                   ; load and check boot sector
        .globl  _rwabs                  ; read/write on block dev
        .globl  _getbpb                 ; get bios parameter block
        .globl  _dskinit                ; disk system initialization
        .globl  _flopvbl                ; floppy vblank handler
        .globl  _floprd                 ; read sector(s)
        .globl  blink                   ; cursor blink (vblank)




*-----
* Default System Parameters.
* Do not change these much.
*-----
df_seek         equ     $0003           ; default seek-rate (3ms)
dnvbls          equ     8               ; default number of vbl queue entries
nlevels         equ     5               ; max # recursive BIOS calls
```

```
savsiz          equ     23              ; size (.W) of BIOS trap save-context



*-------------- Magic Numbers
resmagic        equ     $31415926       ; validates 'resvalid'
diagmagic       equ     $fa52235f       ; validate diagnostic cartridge
apmagic         equ     $abcdef42       ; validate application cartridge
memmagic        equ     $752019f3       ; validates 'memvalid'
memmag2         equ     $237698aa       ; validates 'memval2'
bootmagic       equ     $1234           ; magic checksum for boot sector




*-------------- Data Structures

*---- Floppy state variables:
dfused          equ     0               ; nonzero: floppy has been accessed
dcurtrack       equ     dfused+2        ; current track#
dseekrt         equ     dcurtrack+2     ; floppy's seek-rate

*---- Cartridge application:
ca_next         equ     0               ; (.L) link to next application
ca_flags        equ     4               ; (.B) run flags (MSB of ca_init)
ca_init         equ     4               ; (.L) pointer to init code
ca_run          equ     8               ; (.L) pointer to run code
ca_time         equ     $c              ; (.W) DOS-format creation time
ca_date         equ     $e              ; (.W) DOS-format creation date
ca_size         equ     $10             ; (.L) application size
ca_name         equ     $14             ; application name (NNNNNNNN.EEE\0)


*------ Ram configuration equates
bank1           equ $200000             ; address of 2Mb second bank
twomb           equ 1024*2048           ; two megabytes
one28           equ $20000              ; 128K


*-------------- Hardware Equates

*------ ROM addresses:
romstart        equ $fa0000             ; lowest ROM address
romend          equ $ff0000             ; first byte not in ROM
cartbase        equ $fa0000             ; start of cartridge ROM
cartsize        equ $20000              ; size of cartridge (128K)

*--- Shifter:
memconf         equ     $ffff8001       ; memory controller
syncmode        equ     $ffff820a       ; video sync mode
```

```
dbasel          equ     $ffff8203       ; display base low
dbaseh          equ     $ffff8201       ; display base high
color0          equ     $ffff8240       ; color palette #0
shiftmd         equ     $ffff8260       ; video shift mode (resolution)

*--- DMA chip:
diskctl         equ     $ffff8604       ; disk controller data access
fifo            equ     $ffff8606       ; DMA mode control
dmahigh         equ     $ffff8609       ; DMA base high
dmamid          equ     $ffff860b       ; DMA base medium
dmalow          equ     $ffff860d       ; DMA base low

*--- 1770 select values:
cmdreg          equ     $80             ; 1770/FIFO command register select
trkreg          equ     $82             ; 1770/FIFO track register select
secreg          equ     $84             ; 1770/FIFO sector register select
datareg         equ     $86             ; 1770/FIFO data register select

*--- GI ("psg") sound chip:
giselect        equ     $ffff8800       ; (W) sound chip register select
giread          equ     $ffff8800       ; (R) sound chip read-data
giwrite         equ     $ffff8802       ; (W) sound chip write-data
gimixer         equ     7               ; I/O control/volume control register
giporta         equ     $e              ; GI register# for I/O port A
giportb         equ     $f              ; Centronics output register

*------ Bits in "giporta":
xrts    equ     8                       ; RTS output
dtr     equ     $10                     ; DTR output
strobe  equ     $20                     ; Centronics strobe output
gpo     equ     $40                     ; "general purpose" output

*--- 68901 ("mfp") sticky chip:
mfp     equ     $fffffa00               ; mfp base
gpip    equ     mfp+1                   ; general purpose I/O
aer     equ     mfp+3                   ; active edge reg
ddr     equ     mfp+5                   ; data direction reg
iera    equ     mfp+7                   ; interrupt enable A & B
ierb    equ     mfp+9
ipra    equ     mfp+$b                  ; interrupt pending A & B
iprb    equ     mfp+$d
isra    equ     mfp+$f                  ; interrupt inService A & B
isrb    equ     mfp+$11
imra    equ     mfp+$13                 ; interrupt mask A & B
imrb    equ     mfp+$15
vr      equ     mfp+$17                 ; interrupt vector base
tacr    equ     mfp+$19                 ; timer A control
tbcr    equ     mfp+$1b                 ; timer B control
tcdcr   equ     mfp+$1d                 ; timer C & D control
tadr    equ     mfp+$1f                 ; timer A data
tbdr    equ     mfp+$21                 ; timer B data
tcdr    equ     mfp+$23                 ; timer C data
tddr    equ     mfp+$25                 ; timer D data
scr     equ     mfp+$27                 ; sync char
ucr     equ     mfp+$29                 ; USART control reg
rsr     equ     mfp+$2b                 ; receiver status
```

```
tsr      equ     mfp+$2d                      ; transmit status
udr      equ     mfp+$2f                      ; USART data

*--- 6850 registers:
keyctl   equu    $fffffc00                    ; keyboard ACIA control
keybd    equ     keyctl+2                     ; keyboard data
midictl  equ     $fffffc06                    ; MIDI ACIA control
midi     equ     midictl+2                    ; MIDI data




*+
* Dump area
* Processor state is dumped here after an uncaught trap
*
*-
proc_lives      equ     $380                  ; lives if $12345678
proc_regs       equ     proc_lives+4          ; D0-D7/A0-A7
proc_pc         equ     proc_regs+$40         ; PC
proc_usp        equ     proc_pc+4             ; USP
proc_stk        equ     proc_usp+4            ; six words of stack



*+
* Base of system BSS.
* Starts at $400, just above interrupt vector RAM.
*
* These will never change in future releases of the system.
*
*-
                bss

* "extended" trap vectors:
etv_timer:      ds.l    1       ; (400) vector for timer interrupt chain
etv_critic:     ds.l    1       ; (404) vector for critical error chain
etv_term:       ds.l    1       ; (408) vector for process terminate
etv_xtra:       ds.l    5       ; (40c) 5 reserved vectors

memvalid:       ds.l    1       ; (420) indicates system state on RESET
memcntlr:       ds.w    1       ; (424) mem controller config nibble
resvalid:       ds.l    1       ; (426) validates 'resvector'
resvector:      ds.l    1       ; (42a) [RESET] bailout vector
phystop:        ds.l    1       ; (42e) physical top of RAM
_membot:        ds.l    1       ; (432) bottom of available memory;
_memtop:        ds.l    1       ; (436) top of available memory;
memval2:        ds.l    1       ; (43a) validates 'memcntlr' and 'memconf'
flock:          ds.w    1       ; (43e) floppy disk/FIFO lock variable
seekrate:       ds.w    1       ; (440) default floppy seek rate
_timr_ms:       ds.w    1       ; (442) system timer calibration (in ms)
_fverify:       ds.w    1       ; (444) nonzero: verify on floppy write
_bootdev:       ds.w    1       ; (446) default boot device
palmode:        ds.w    1       ; (448) nonzero ==> PAL mode
defshiftmd:     ds.w    1       ; (44a) default video rez (first byte)
```

```
sshiftmd:       ds.w    1            ; (44c)  shadow for 'shiftmd' register
_v_bas_ad:      ds.l    1            ; (44e)  pointer to base of screen memory
vblsem:         ds.w    1            ; (452)  semaphore to enforce mutex in  vbl
nvbls:          ds.w    1            ; (454)  number of deferred vectors
_vblqueue:      ds.l    1            ; (456)  pointer to vector of deferred  vfuncs
colorptr:       ds.l    1            ; (45a)  pointer to palette setup (or NULL)
screenpt:       ds.l    1            ; (45e)  pointer to screen base setup (|NULL)
_vbclock:       ds.l    1            ; (462)  count  of unblocked vblanks
_frclock:       ds.l    1            ; (466)  count  of every vblank

hdv_init:       ds.l    1            ; (46a)  hard disk initialization
swv_vec:        ds.l    1            ; (46e)  video change-resolution bailout
hdv_bpb:        ds.l    1            ; (472)  disk "get BPB"
hdv_rw:         ds.l    1            ; (476)  disk read/write
hdv_boot:       ds.l    1            ; (47a)  disk "get boot sector"
hdv_mediach:    ds.l    1            ; (47e)  disk media change detect

_cmdload:       ds.w    1            ; (482)  nonzero: load COMMAND.COM from boot
conterm:        ds.b    1            ; (484)  console/vt52 bitSwitches (%%0..%%2)
                ds.b    1            ; (485)  [unused, reserved]

trpl4ret:       ds.l    1            ; (486)  saved return addr for _trap14
criticret:      ds.l    1            ; (48a)  saved return addr for _critic
themd:          ds.l    4            ; (48e)  memory descriptor (MD)
______md:       ds.w    2            ; (49e)  (more MD)
savptr:         ds.l    1            ; (4a2)  pointer to register save area

_nflops:        ds.w    1            ; (4a6)  number of disks attached (0, 1+)
con_state:      ds.l    1            ; (4a8)  state of conout() parser
save_row:       ds.w    1            ; (4ac)  saved row# for cursor X-Y addressing
sav_context:    ds.l    1            ; (4ae)  pointer to saved processor context
_bufl:          ds.l    2            ; (4b2)  two buffer-list headers
_hz_200:        ds.l    1            ; (4ba)  200hz raw system timer tick
                ds.l    1            ; (4be)  reserved for future use
_drvbits:       ds.l    1            ; (4c2)  bit vector of "live" block devices
_dskbufp:       ds.l    1            ; (4c6)  pointer to common disk buffer
_autopath:      ds.l    1            ; (4ca)  pointer to autoexec path (or NULL)
_vbl_list:      ds.l    8            ; (4ce)  initial _vblqueue (to $4ee)
_prtcnt:                             * (4ee)  screen-dump flag alias
_dumpflg:       ds.w    1            ; (4ee)  screen-dump flag
_prtabt:        ds.w    1            ; (4f0)  printer abort flag
_sysbase:       ds.l    1            ; (4f2)  -> base of OS
_shell_p:       ds.l    1            ; (4f6)  -> global shell info
end_os:         ds.l    1            ; (4fa)  -> end of OS memory usage
exec_os:        ds.l    1            ; (5fe)  -> address of shell to exec on startup

* Start of no-man's land (locations beyond this point subject to change):
the_env:        ds.b    20                     ; space for a small enviroment string
savarea:        ds.w    savsiz*nlevels         ; register save area
savend:                                        * end of register sav area
endosbss:                                      * end of "base" BSS
```

```
        .text
*+
* System startup parameters
*
* In ROM, these are found at $FC0000.
* In any event, they are found at *(_st_begos).
*
*-
ostext:         bra.s   reseth          ; ($0)  branch to reset handler
                dc.w    version         ; ($2)  OS version number
                dc.l    reseth          ; ($4)  -> system reset handler
os_beg:         dc.l    ostext          ; ($8)  -> base of OS
os_end:         dc.l    endos           ; ($c)  -> end of OS memory usage
os_exec:        dc.l    reseth          ; ($10) -> default shell
os_magic:       dc.l    the_magic       ; ($14) -> GEM magic (or NULL)
os_date:        dc.l    date            ; ($18) date the system was built
os_conf:        dc.w    0               ; ($1a) configuration bits


 ifeq systype-rom
*+
* [ROM based system]
* reseth - System reset handler
*
*  Gains control of the system upon power-up reset,
*  or when the RESET button is pressed,
*  or after a really messy system crash.
*
*-
reseth:
        move.w  #$2700,sr               ; super mode, no interrupts
        reset                           ; reset hardware
 endc


 ifeq systype-ram
*+
* [RAM based system]
* reseth - Startup the system
*
*  Gains control from the boot loader
*  as soon as the OS has been relocated.
*
*-
reseth:
        move.w  #$2700,sr               ; super mode, no interrupts
 endc


 ifeq systype-rom
*+
*  [ROM based system]
*  Check for a diagnostic cartridge;
*  if one is inserted, load a return address
*  into A6 and jump to the cart's entry point.
```

```
*
*-
        cmp.l   #diagmagic,cartbase     ; is the magic number there?
        bne     reset1                  ; (no)
        lea     reset1(pc),a6           ; a6 -> return address
        jmp     cartbase+4              ; execute diagnostic cartridge
 endc


 ifeq systype-rom
*+
*  [ROM based system]
*  If this is a warm reset, setup the memory
*  controller configuration register so that
*  the reset-bailout vector has something to
*  stand on ....
*
*-
reset1:
        lea     ret_1(pc),a6            ; load return addr
        bra     val_memval              ; check memory configuration validity
ret_1:  bne     reset2                  ; (invalid -- don't set anything up)
        move.b  memcntlr,memconf        ; initialize memory controller
 endc


 ifeq systype-rom
*+
*  [ROM based system]
*  RESET bailout vector check.
*  Check to make sure we have a clean, well-bred
*  bailout vector.  The high byte must be zero,
*  it must be even, and cannot be entirely zero.
*
*-
reset2: clr.l   a5                      ; quick zeropage
        cmp.l   #resmagic,resvalid(a5)  ; is resvalid the magic number?
        bne     reset3                  ; (no)
        move.l  resvector(a5),d0        ; d0 = reset bailout vector
        tst.b   resvector(a5)           ; bits 24..31 must be zero
        bne     reset3                  ; (they aren't, so punt)
        btst    #0,d0                   ; the vector must be even
        bne     reset3                  ; (it isn't, so punt)
        move.l  d0,a0                   ; a0 -> reset handler
        lea     reset2(pc),a6           ; a6 -> return address
        jmp     (a0)                    ; execute reset bailout
 endc


 ifeq systype-ram
*+
*  [RAM based system]
*  Setup the reset-bailout vector to point
*  to our own system-reset handler.
*
*-
```

```
        move.l  reseth,resvector
        move.l  #resmagic,resvalid
 endc


*+
*  Initialize PSG output ports.
*  Make ports A and B output-only;
*  initialize floppy select lines (so
*  that none are selected).
*
*-
reset3: lea     giselect,a0                     ; a0 -> giselect, giwrite-2
        move.b  #7,(a0)                         ; set porta & portb to output
        move.b  #$c0,2(a0)
        move.b  #$e,(a0)                        ; deselect disks
        move.b  #7,2(a0)


*+
*  Determine 50hz or 60hz.
*  The hardware RESETs to 60hz.  Check a bit in the
*  ROM configuration byte to see if we have to twiddle
*  the hardware into 50hz mode.
*
*-
        btst.b  #0,os_conf(pc)                  ; check bit: configured for 50hz?
        beq     notpal                          ; (nope -- we're good ol' NTSC)
        move.b  #$02,syncmode                   ; yes -- twiddle to 50hz
notpal:


*+
*  Initialize palette registers to
*  their default values.
*
*-
        lea     color0,a1                       ; a1 -> hardware reg
        move.w  #16-1,d0                        ; setup 16 colors
        lea     colors(pc),a0                   ; a0 -> table of default colors
sysic1: move.w  (a0)+,(a1)+                     ; copy palette assignment
        dbra    d0,sysic1                       ; (loop for more colors)


 ifeq systype-rom
*+
*  On a ROM system, put the screen (temporarily)
*  at $10000, so the icon-drawing routines won't
*  blow away any system variables.
*
*-
        move.b  #$01,dbaseh                     ; set high ptr
        move.b  #$00,dbasel                     ; set low ptr
 endc
 ifeq systype-rom
*+
*  [ROM based system]
*  Determine how much memory there is, and initialize
*  the memory controller configuration register.
*
*  Algorithm from Jim Tittsler, Art Morgan, et al.
*  but shamelessly modified for the hell of it.
*
*  The bottom 1K of memory is only touched on the first RESET,
*  to size memory and setup the memory controller.  The first 1K
*  is never cleared.
*
*-
        clr.l   a5                      ; quick zeropage
        move.b  memcntlr(a5),d6         ; d6 = memory controller configuration
        move.l  phystop(a5),d5          ; d5 -> (possible) top of physical mem
        lea     ret_2(pc),a6            ; load return address
        bra     val_memval              ; get memory controller validation
ret_2:  beq     reset4                  ; already sized -- don't size or test

*---- init vars + hardware:
        clr.w   d6                      ; d6 = configuration byte
        clr.l   d5                      ; d5 -> physical top of RAM
        move.b  #$0a,memconf            ; setup controller for 2Mb/2Mb

*---- write test-pattern to both banks:
        move.w  #8,a0                   ; a0 -> bank0 (skip ROM shadow)
        lea     bank1+8,a1              ; a1 -> bank1
        clr.w   d0                      ; d0 = start of pattern
fmem1:  move.w  d0,(a0)+                ; write to bank 0
        move.w  d0,(a1)+                ; write to bank 1
        add.w   #$fa54,d0               ; bump pattern with a magic number
        cmp.l   #$200,a0                ; filled $200 bytes?
        bne     fmem1                   ; (no, loop)

*+
* Determine size of both banks
* from test-pattern signatures:
*-
        move.l  #bank1,d1               ; d1 = bank offset (start with bank 1)
mem1:   lsr.w   #2,d6                   ; (shift bank1's size into position)
        move.w  #$208,a0                ; pattern matches at $208?
        lea     memr1(pc),a5            ; a5 -> return addr
        bra     memchk                  ; (check the pattern)
memr1:  beq     mem2                    ; yes -- 128K
        move.w  #$408,a0                ; pattern matches at $408?
        lea     memr2(pc),a5            ; a5 -> return addr
        bra     memchk                  ; (check it)
memr2:  beq     mem3                    ; yes -- 512K
        move.w  #8,a0                   ; pattern matches at $8?
        lea     memr3(pc),a5            ; a5 -> return addr
        bra     memchk                  ; (attempt match)
memr3:  bne     mem4                    ; no -- nothing in this bank
        add.l   #bank1-$80000-$20000,d5 ; adjust size for 2M bank
        addq.w  #4,d6                   ; adjust config byte for 2M
```

```
mem3:   add.l   #$80000-$20000,d5       ; adjust size for 512K bank
        addq.w  #4,d6                   ; adjust config byte for 512K
mem2:   add.l   #$20000,d5              ; adjust size for 128K bank
mem4:   sub.l   #bank1,d1               ; decrement bank number
        beq     mem1                    ; repeat check for bank 0
cold3:  move.b  d6,memconf              ; setup memory controller
 endc


 ifeq systype-rom
*+
*  [ROM based system]
*  Clear memory from $400 to 'd5' (phystop).
*
*-
        move.l  d5,a0                   ; start at the end
        move.l  #$400,d4                ; where to end
        movem.l zeros(pc),d0-d3         ; get some cheap zeros
clm_1:  movem.l d0-d3,-(a0)             ; ... work our way back
        cmp.l   d4,a0                   ; done?
        bne     clm_1                   ; (loop for more bytes)
 endc


 ifeq systype-rom
*+
*  Indicate that memory has successfully
*  been sized and tested.  Set two variables
*  to magic values ...
*
*-
        clr.l   a5                      ; cheap zeropage
        move.b  d6,memcntlr(a5)         ; save configuration byte
        move.l  d5,phystop(a5)          ; save physical top-of-memory
        move.l  #memmagic,memvalid(a5)  ; indicate memory was configured
        move.l  #memmag2,memval2(a5)    ; ditto (paranoia variable)
 endc


reset4:
        clr.l   a5                      ; quick zeropage


 ifeq systype-rom
*+
*  [ROM system]
*  Clear bottom 64K (or so) of memory.
*  (this is sufficient for GEMDOS and the AES,
*   which require their BSS to be zero when
*   they are started up).
*
*-
        move.w  #endosbss,a0            ; a0 -> start
        move.l  #$10000,a1              ; a1 -> end
 endc
 ifeq systype-ram
*+
*  [RAM loaded system]
*  Clear OS bss (from 'endosbss' to 'ostext')
*
*-
        move.w  #endosbss,a0                    ; a0 -> start
        move.w  #ostext,a1                      ; a1 -> end
 endc


*---- common code to clear memory:
        moveq   #0,d0                           ; quick zero
clrm_1: move.w  d0,(a0)+                        ; clobber a word
        cmp.l   a0,a1                           ; at end?
        bne     clrm_1                          ; (no -- loop for more words)


*+
*  Setup display base,
*  clear display memory.
*
*-
        move.l  phystop(a5),a0                  ; video_base = phystop - 0x8000
        sub.l   #$8000,a0
        move.l  a0,_v_bas_ad(a5)
        move.b  _v_bas_ad+1(a5),dbaseh          ; load high addr
        move.b  _v_bas_ad+2(a5),dbasel          ; load low (really, medium) addr
        move.w  #$800-1,d1                      ; d1 = # 16-byte chunks to zero
clrm_2: move.l  d0,(a0)+                        ; zero a longword
        move.l  d0,(a0)+                        ; zero a longword
        move.l  d0,(a0)+                        ; zero a longword
        move.l  d0,(a0)+                        ; zero a longword
        dbra    d1,clrm_2                       ; (loop for more longwords)


*+
*  Initialize all kinds of OS variables
*
*-

*---- OS parameters:
        move.l  os_magic(pc),a0                 ; get pointer to magic
        cmp.l   #$87654321,(a0)                 ; is the magic there?
        beq     usem                            ; yes -- use numbers there
        lea     os_end-4(pc),a0                 ; no, use default numbers
usem:   move.l  4(a0),end_os                    ; init end-of-OS pointer
        move.l  8(a0),exec_os                   ; init default-shell pointer

*---- Disk vectors:
        move.l  #_dskinit,hdv_init(a5)          ; initialization
        move.l  #_rwabs,hdv_rw(a5)              ; read/write absolute sectors
        move.l  #_getbpb,hdv_bpb(a5)            ; get BIOS parameter block
        move.l  #_mediach,hdv_mediach(a5) ; media change inquiry
        move.l  #_boot,hdv_boot(a5)       ; boot-from-device

*--- Randoms:
        move.l  _v_bas_ad(a5),_memtop(a5) ; _memtop = _v_bas_ad
        move.l  end_os(a5),_membot(a5)   ; set bottom of memory (for DOS)
        lea     _supstk+2048,sp          ; setup supervisor stack
        move.w  #dnvbls,nvbls(a5)        , default number of vbl queue entries
        st      _fverify(a5)             ; enable write-verify
        move.w  #df_seek,seekrate(a5)    ; set default seek-rate
        move.l  #_diskbuf,_dskbufp(a5)   ; setup pointer to disk buffer
        move.w  #-1,_prtcnt(a5)          ; initialize print-count
        move.l  #ostext,_sysbase(a5)     ; -> base of OS
        move.l  #savend,savptr(a5)       ; register-save pointer for traps 13&14
        move.l  #_rts,swv_vec(a5)        ; ignore monitor changes for now


*+
*  Initialize interrupt vectors
*
*  If a diagnostic cartridge is inserted, the "random" vectors
*  (for Bus Error, Address Error, and so on) are left alone.
*
*  Otherwise, the random vectors are pointed to the system critical
*  error handler (_term).  The high byte of the vector (bits 24..31)
*  contains the exception number.  [Yes, this will lose on a 68020.]
*
*  Trap 2 and Divide-by-zero are pointed at an RTE.
*
*  The HBLANK, VBLANK, line 1010, [someday: line 1111], trap 13, trap 14,
*  and "extended" trap vectors are initialized appropriately.
*
*-
        lea     _rte(pc),a3              ; a3 -> handy RTE
        lea     _rts(pc),a4              ; a4 -> handy RTS

*--- diagnostic cartridge check:
        cmp.l   #diagmagic,cartbase      ; check cartridge magic
        beq     sei2                     ; (it's there -- leave vectors alone)

*--- setup 62 vectors:
        lea     _term(pc),a1             ; a1 -> "terminate process" handler
        add.l   #$02000000,a1            ; a1 += vector number (high byte)
        lea     8,a0                     ; a0 -> interrupt RAM
        move.w  #64-2-1,d0               ; d0 = count
sei1:   move.l  a1,(a0)+                 ; write vector
        add.l   #$01000000,a1            ; bump vector number in bits 24..31
        dbra    d0,sei1                  ; (loop to write more vectors)
        move.l  a3,$14                   ; divide-by-zero vector -> rte

*--- install OS interrupt vectors:
sei2:   move.l  #vbl,$70(a5)             ; vblank handler
        move.l  #hbl,$68(a5)             ; hblank handler
        move.l  a3,$88(a5)               ; (empty) trap#2 handler
        move.l  #trp13h,$b4(a5)          ; trap #13 handler
        move.l  #trp14h,$b8(a5)          ; trap #14 handler
        move.l  #line1010,$28(a5)        ; line 1010 handler
        move.l  a4,etv_timer(a5)          ; default timer-tick vector -> rts
        move.l  #_critich,etv_critic(a5) ; default critical error handler
        move.l  a4,etv_term(a5)           ; default terminate vector -> rts


*+
*  Setup the vblank deferred vector list.
*  (This data structure is ugly,
*   but we seem to be stuck with it).
*
*-
        lea     _vbl_list(a5),a0          ; a0 -> default list of vbl locs
        move.l  a0,_vblqueue(a5)          ; install ptr to them
        move.w  #dnvbls-1,d0              ; clear vbl vectors
avbl:   clr.l   (a0)+                     ;       one at a time
        dbra    d0,avbl


*+
*  "The other half" of the BIOS handles character I/O;
*  call its initialization hook.
*  (It can "never fail".  This will get interesting
*   if we ever do a detachable keyboard ....)
*
*-
        bsr     initmfp


*+
*  Fire up %%2 cartridges
*
*-
        moveq   #2,d0                     ; bit# = 2
        bsr     cartscan                  ; execute cartridge aps


*+
*  Initialize screen resolution,
*  kludge color lookup RAM for medium-rez (if we're in it).
*
*-
        clr.l   a5                        ; quick zero page (again)
        bsr     wvbl                      ; flush pending VBI
        bsr     wvbl                      ; wait for next VBI
        move.b  #2,d0                     ; assume high-rez monitor
        btst.b  #7,gpip                   ; test "HighRez" panic input
        beq     setvbl                    ; (set high-resolution)
        move.b  defshiftmd(a5),d0         ; get default color mode
        cmp.b   #2,d0                     ; if(mode >= 2) mode = 0
        blt     setvbl
        clr.b   d0
setvbl: move.b  d0,sshiftmd(a5)           ; set rez shadow
        move.b  d0,shiftmd                ; set rez hardware register

*--- if in medium rez, hack color3 := color15 (for GSX)
        cmp.b   #1,d0                     ; in medium rez?
        bne     setvb2                  ; (no, so don't fiddle)
        move.w  color0+$1e,color0+6     ; copy color 15 to color 3

setvb2: jsr     esc_init                ; clear screen, initialize cursor
        move.l  #reseth,swv_vec(a5)     ; RESET system on monitor change
        move.w  #1,vblsem               ; enable vblank processing


*+
*  [1] Fire up %%0 cartridges;
*  [2] Enable interrupts;
*  [3] Fire up %%1 cartridges
*
*-
        clr.w   d0                      ; magic bit# = 0
        bsr     cartscan                ; execute cartridge aps
        move.w  #$2300,sr               ; go to IPL 3
        moveq   #1,d0                   ; magic bit# = 1
        bsr     cartscan                ; execute cartridge aps


*+
* Load shell (if _cmdload is nonzero)
* or execute GEM in ROM
*
*-
        bsr     _osi                    ; initialize DOS
        bsr     _dskboot                ; attempt to boot from disk
        tst.w   _cmdload                ; load shell from disk?
        beq     st_1                    ; (no -- execute GEM in ROM)

        bsr     esce                    ; turn on cursor
        bsr     _auto                   ; do auto-exec

        pea     nullenv(pc)             ; null enviroment string
        pea     nullenv(pc)             ; null argument string
        pea     cmdname(pc)             ; push shell filename
        clr.w   -(sp)                   ; load-and-go flavor of exec
        bra     st_x                    ; exec shell ("never return")

*--- bring up GEM:
st_1:   bsr     _auto                   ; do auto-exec

*--- kludge up an enviroment string:
        lea     orig_env(pc),a0         ; a0 -> original enviroment string
        move.w  #the_env,a1             ; a1 -> place to put it
st_2:   cmp.b   #'#',(a0)               ; look for drive# character
        bne     st_3                    ; (not it)
        move.l  a1,a2                   ; a2 -> place to put drive#
st_3:   move.b  (a0)+,(a1)+             ; copy a byte
        bpl     st_2                    ; loop while not end-of-string

        move.b  _bootdev,d0             ; compute drive#, and shove it
        add.b   #'A',d0                 ; into the env string at the
        move.b  d0,(a2)                 ; appropriate spot
```

```
* kludge up an enviroment string:
        pea     the_env                 ; push address of enviroment string
        pea     nullenv                 ; no arguments

* ifeq systype-ram
*       pea     gemname(pc)             ; exec GEM.PRG
*       clr.w   -(sp)                   ; load-and-go
* endc

* ifeq ramloaded
        pea     nullenv(pc)             ; null shell name (in ROM, after all)
        move.w  #5,-(sp)                ; createPSP flavor of exec
        move.w  #$4b,-(sp)              ; exec function#
        trap    #1                      ; get pointer to PSP
        add.w   #14,sp                  ; (clean up cruft)
        move.l  d0,a0                   ; a0 -> PSP
        move.l  exec_os,8(a0)           ; stuff saddr of GEM in PSP

        pea     the_env                 ; our enviroment string
        move.l  a0,-(sp)                ; push addr of PSP
        pea     nullenv(pc)             ; null filename
        move.w  #4,-(sp)                ; just-go
* endc

st_x:   move.w  #$4b,-(sp)              ; function = exec
        trap    #1                      ; do it
        add.w   #14,sp                  ; cleanup stack

*+
* When startup fails (or if the exec returns,
* which "cannot happen") fake a system reset:
*-
        jmp     reseth                  ; back to the beginning...


*+
* Default enviroment string
* Cannot be more than 20 chars long without modifying
* the declaration for the_env;
* Any char >= $80 terminates the string (and is included in it)
* The last '#' character is replaced by the boot drive's name (A, B, ...)
*-
orig_env: dc.b  "PATH=",0               ; default pathname
          dc.b  "#:\",0                 ; is the boot device
          dc.b  0                       ; terminate env string
          dc.b  $ff                     ; end of env string (for our copy)


cmdname: dc.b   "COMMAND.PRG",0         ; shell name
gemname: dc.b   "GEM.PRG"               ; desktop name
nullenv: dc.b   0,0                     ; null string (and enviroment)
        even


*+
* _dskboot - boot (or return diagnostics)
* Passed:        nothing
* Returns:       D0.W = error number (if nonzero)
*-
_dskboot:
        moveq   #3,d0                   ; %%3 ap cart
        bsr     cartscan
        move.l  hdv_boot,a0             ; go through boot vector
        jsr     (a0)
 ifeq systype-rom
        tst.w   d0                      ; any errors?
        bne     dskb1                   ; (yes -- punt)
        lea     _diskbuf,a0             ; a0 -> disk buffer
        jsr     (a0)                    ; execute boot sector (it might return)
 endc
dskb1:  rts                             ; return status


*+
* cartscan - scan cartridge memory for runable applications
* Passed:        d0 = bit# to test in application's initialization vector
* Returns:       after all applications have been examined
* Uses:          a0,d0
*-
cartscan:
        lea     cartbase,a0             ; a0 -> cartridge memory
        cmp.l   #apmagic,(a0)+          ; correct magic number?
        bne     ca_r                    ; (no, so return)

ca_1:   btst.b  d0,ca_flags(a0)         ; test bit in MSB of INIT address
        beq     ca_2                    ; (not set, so don't execute)
        movem.l d0-d7/a0-a6,-(sp)       ; save everything
        move.l  ca_init(a0),a0          ; a0 -> initialization address
        jsr     (a0)                    ; call cartridge application
        movem.l (sp)+,d0-d7/a0-a6       ; restore everything
ca_2:   tst.l   (a0)                    ; test link address
        move.l  (a0),a0                 ; a0 -> next header (or NULL)
        bne     ca_1                    ; loop on next header
ca_r:   rts


_rts:   rts


*+
* memchk - check pattern written to memory
*       Passed:         d1.l = offset
*                       a0 = base of pattern ($1f8 bytes long)
*                       a5 -> return address
*
*       Returns:        EQ: the pattern matched
*                       NE: the pattern didn't match
*
*       Uses:           d0.w, a1
*
```

```
*       Called-by:      Coldstart memory-sizing routine.
*-
memchk:
        add.l   d1,a0                   ; a0 -> memory to check
        clr.w   d0                      ; zap pattern seed
        lea     $1f8(a0),a1             ; a1 -> ending address
memchk1: cmp.w  (a0)+,d0                ; match?
        bne     memchkr                 ; (no -- return NE)
        add.w   #$fa54,d0               ; yes -- bump pattern
        cmp.l   a0,a1                   ; matched entire pattern?
        bne     memchk1                 ; (no)
memchkr: jmp    (a5)                    ; "return" to caller



 ifeq systype-rom
*+
* sysfail - we drop dead gracefully (sort of)
*
*  If on a high-rez system, set video configuration to high-rez;
*  Put up some diagnostic info;
*  Display some kind of icon in the screen's center;
*  Then loop forever, incrementing a bit of screen memory ....
*
*-
sysfail:
        btst.b  #7,gpip                 ; test "HighRez" panic input
        bne     sysf1                   ; (keep low rez)
        move.b  #$02,shiftmd            ; set high rez, cross our fingers

sysf4:  lea     sysf1(pc),a6            ; load return address
        lea     failure(pc),a1          ; a1 -> icon form
        bra     sysfdraw                ; draw icon
sysf1:  moveq   #0,d0                   ; delay a while
sysf5:  dbra    d0,sysf5
        lea     sysf2(pc),a6            ; load return address
        lea     failure1(pc),a1         ; a1 -> icon form
        bra     sysfdraw                ; draw it
sysf2:  moveq   #0,d0                   ; delay a while
sysf3:  dbra    d0,sysf3
        bra     sysf4                   ; back to the beginning ....

sysfdraw:
        clr.l   a0                      ; draw in middle of screen
        moveq   #0,d7                   ; count = 1
        lea     failure(pc),a1          ; a1 -> icon form
        bra     _draw_icon              ; draw the icon


*+
*  "Sad" icon form
*  ... or something like that ....
*
*-
failure:
        dc.w    %1111111111111111
        dc.w    %1000000000000001
        dc.w    %1000001111000001
        dc.w    %1000001111000001
        dc.w    %1000001111000001
        dc.w    %1000001111000001
        dc.w    %1000001111000001
        dc.w    %1000001111000001
        dc.w    %1000001111000001
        dc.w    %1000001111000001
        dc.w    %1000001111000001
        dc.w    %1000000000000001
        dc.w    %1000001111000001
        dc.w    %1000001111000001
        dc.w    %1000000000000001
        dc.w    %1111111111111111

*--- alternate form of the thing:
failure1:
        dc.w    %1111111111111111
        dc.w    %1000000000000001
        dc.w    %1000000000000001
        dc.w    %1000000000000001
        dc.w    %1000000000000001
        dc.w    %1000000000000001
        dc.w    %1000000000000001
        dc.w    %1000000000000001
        dc.w    %1000000000000001
        dc.w    %1000000000000001
        dc.w    %1000000000000001
        dc.w    %1000000000000001
        dc.w    %1000000000000001
        dc.w    %1000000000000001
        dc.w    %1000000000000001
        dc.w    %1111111111111111
 endc


*+
* val_memval - test memory configuration validation
*  Passed:      a6 -> return addressd
*  Returns:     a5 -> 0 (quick zeropage)
*               EQ: memory setup OK
*               NE: memory never configured succesfully
*
*-
val_memval:
        clr.l   a5                      ; a5 -> quick zeropage
        cmp.l   #memmagic,memvalid(a5)  ; check first magic number
        bne     val_mr                  ; (mismatched -- return NE)
        cmp.l   #memmag2,memval2(a5)    ; check once more (for paranoia)
val_mr: jmp     (a6)                    ; return EQ/NE


*+
*  Four longwords of zero
```

```
*
*-
zeros:   dc.l     $00000000,$00000000,$00000000,$00000000


*+
* _draw_icon - Draw an icon
*  Passed:        a6 -> return address
*                 a1 -> source form
*                 d5 = #icons to draw - 1
*                 a0 = destination:
*                         a0 == 0:        draw in middle of screen
*                         a0 < $8000:     draw at offset on screen
*                         a0 >= $8000:    draw in memory
*
*  Uses:          d0-d7/a0-a3/a5
*
*-
_draw_icon:
        move.b  shiftmd,d4
        and.w   #$0003,d4
        add.w   d4,d4                       ; d4 = rez index

        cmp.l   #$8000,a0                   ; if (a0 >= 0x8000) just_use_it;
        bhi     di_na
        cmp.l   #0,a0                       ; if (a0 == 0) a0 = icn_index[d4]
        bne     di_na1
        move.w  icn_index(pc,d4.w),a0       ; get offset of middle of screen
di_na1: clr.l   d0                          ; d0 = base_of_screen
        move.b  dbaseh,d0
        lsl.w   #8,d0
        move.b  dbasel,d0
        lsl.l   #8,d0
        add.l   d0,a0                       ; a0 += base_of_screen;

di_na:  moveq   #15,d7                      ; d7 = scanline count
di_1:   move.w  icn_repeat(pc,d4.w),d6      ; d6 = #scanlines to repeat
di_2:   move.w  d5,d3                       ; d3 = count of # to draw
        move.l  a0,a2                       ; a2 -> next scanline
        add.w   icn_width(pc,d4.w),a2
di_3:   move.b  (a1),d0                     ; get word from source form
        lea     di_rt1(pc),a5               ; (a5->return address)
        bra     dup8                        ; expand MSW of icon
di_rt1: move.w  d2,d1
        move.b  1(a1),d0                    ; expand LSW of icon
        lea     di_rt2(pc),a5               ; (a5->return address)
        bra     dup8
di_rt2: move.w  (a1),d0                     ; get original icon word
        jmp     di_jmp(pc,d4.w)             ; jump to draw routine

di_jmp: bra.s   di_low
        bra.s   di_med
        bra.s   di_hi

di_low: move.w  d0,(a0)+                    ; store all four planes in lorez
```

```
        move.w  d0,(a0)+
        move.w  d0,(a0)+
        move.w  d0,(a0)+
        bra.s   di_cn                   ; (continue)

di_med: move.w  d1,(a0)+                ; store plane 0
        move.w  d1,(a0)+                ; store plane 1
        move.w  d2,(a0)+                ; store plane 0
        move.w  d2,(a0)+                ; store plane 1
        bra.s   di_cn                   ; (continue)

di_hi:  move.w  d1,(a0)+                ; store plane 0
        move.w  d2,(a0)+                ; store plane 1

di_cn:  dbra    d3,di_3                 ; loop to do more on this line
        move.l  a2,a0                   ; a0 -> next scanline
        dbra    d6,di_2                 ; dup scanlines
        addq    #2,a1                   ; bump source form
        dbra    d7,di_1                 ; do another scanline
        jmp     (a6)                    ; return


*+
* dup8 - expand d0.b into d2.w
*  Passed:      d0.b = source bits
*               a5 -> return address
*
*  Returns:     d2.w = d0.b, with every bit doubled
*
*  Uses:        a3 (to save d3)
*
*-
dup8:   move.l  d3,a3                   ; save d3
        moveq   #0,d2                   ; d2 is pristine
        moveq   #7,d3                   ; d3 = bit count
d8_1:   roxl.b  #1,d0                   ; get MSB into carry + X
        move.w  sr,-(sp)                ; (save X)
        roxl.w  #1,d2                   ; then rotate X in once
        move.w  (sp)+,sr                ; (restore X)
        roxl.w  #1,d2                   ; then rotate X in twice
        dbra    d3,d8_1                 ; (loop for more bits)
        move.l  a3,d3                   ; restore d3
        jmp     (a5)                    ; return

*+
* Screen-rez dependent parameters:
*   o  index to center of screen
*   o  with of screen in bytes
*   o  number of scanlines to repeat
*
*-
icn_index:              dc.w    100*160+72,100*160+72,200*80+36
icn_width:              dc.w    160,160,80
icn_repeat:             dc.w    0,0,1
```

```
*+
* Default palette assignments.
* Sort of corresponding to the GSX spec.
*-
colors: dc.w    $777                    ; 0 white
        dc.w    $700                    ; 1 red
        dc.w    $070                    ; 2 green
        dc.w    $770                    ; 3 yellow
        dc.w    $007                    ; 4 blue
        dc.w    $707                    ; 5 magenta
        dc.w    $077                    ; 6 cyan
        dc.w    $555                    ; 7 "low white"
        dc.w    $333                    ; 8 grey
        dc.w    $733                    ; 9 light red
        dc.w    $373                    ; 10 light green
        dc.w    $773                    ; 11 light yellow
        dc.w    $337                    ; 12 light blue
        dc.w    $737                    ; 13 light magenta
        dc.w    $377                    ; 14 light cyan
        dc.w    $000                    ; 15 black


*+
* hbl - force caller to IPL
* Oh-well:      "Yeah, it sucks, but it works" (--lt)
*
* Note:         Hacks caller's IPL to 3 (if it was 0).  This is
*               a kludge against fascist programs and certain
*               debuggers that insist on starting processes up
*               at IPL 0.
*
*-
hbl:    move.w  d0,-(sp)                ; save d0
        move.w  2(sp),d0                ; get pushed SR
        and.w   #$0700,d0               ; strip crufty bits
        bne     hbl_r                   ; not IPL 0, so punt
        or.w    #$0300,2(sp)            ; force caller to IPL 3
hbl_r:  move.w  (sp)+,d0                ; restore d0, back to victim
        rte


*+
* vbl - vertical blank interrupt handler
*
*-
vbl:
        addq.l  #1,_frclock             ; bump frame clock
        subq.w  #1,vblsem               ; P(vblsem) -- vblank locked?
        bmi     vblret

        movem.l d0-d7/a0-a6,-(sp)       ; save registers
        addq.l  #1,_vbclock             ; bump unblocked-frame clock
        clr.l   a5                      ; a5 -> zero page

*------ Video monitor fail-safe anti-burnout check:
        move.b  shiftmd,d0                      ; get current rez
        and.b   #3,d0                           ; strip bucky bits
        cmp.b   #2,d0                           ; low or high rez?
        bge     swmon1                          ; (high)

*--- low rez: switch to high if gpip%%7 == 0
        btst.b  #7,gpip                         ; get "High rez" input
        bne     swmon3                          ; no change: punt
        move.b  #2,d0                           ; trans to high rez
        bra     swmon2

*--- high rez: switch to low (hopefully defshiftmd) if gpip%%7 == 1
swmon1: btst.b  #7,gpip                         ; get "High rez" input
        beq     swmon3                          ; no change (still highrez)
        move.b  defshiftmd(a5),d0               ; get preferred rez
        cmp.b   #2,d0                           ; if high-rez, then force low rez
        blt     swmon2                          ; (low or med rez)
        clr.b   d0
swmon2: move.b  d0,sshiftmd(a5)                 ; set shadow & hardware shift-mode
        move.b  d0,shiftmd
        move.l  swv_vec(a5),a0                  ; go through "change rez" panic vector
        jsr     (a0)
swmon3:

        bsr     blink                           ; blink cursor

*--- reload color palettes
        clr.l   a5                              ; a5 -> zero page
        tst.l   colorptr(a5)                    ; if(colorptr != NULL)....
        beq     vbl1                            ; (its NULL, so don't reload)
        move.l  colorptr(a5),a0                 ; a0 -> user's color base
        lea     color0,a1                       ; a1 -> hardware palette base
        move.w  #16-1,d0                        ; d0 = count
vbl2:   move.w  (a0)+,(a1)+                     ; load a palette
        dbra    d0,vbl2                         ; ...and repeat
        clr.l   colorptr(a5)                    ; zap colorptr

*--- reload display base register
vbl1:   tst.l   screenpt(a5)                    ; if(screenpt == NULL) don't;
        beq     vbl5
        move.l  screenpt(a5),_v_bas_ad(a5)              ; set OS variable
        move.l  _v_bas_ad(a5),d0                        ; d0 -> screen bottom
        lsr     #8,d0                           ; strip lower 8 bits
        move.b  d0,dbasel                       ; load "low" pointer
        lsr     #8,d0
        move.b  d0,dbaseh                       ; load "high" pointer

*------ Floppy drive-select timeout:
vbl5:   bsr     _flopvbl                        ; (no args)

*------ Call deferred interrupt vectors:
        move.w  nvbls,d7                        ; d7 = # of deferred vblank vectors
        beq     vbl12                           ; (punt if no vectors)
        subq.l  #1,d7                           ; turn into DBRA count
        move.l  _vblqueue,a0                    ; a0 -> vectors
vbl10:  move.l  (a0)+,a1                        ; a1 -> deferred vector
```

```
        cmp.l   #0,a1                   ; if(a1 == NULL) continue;
        beq     vbl11
        movem.l d7/a0,-(sp)             ; save registers
        jsr     (a1)                    ; call routine
        movem.l (sp)+,d7/a0             ; restore registers
vbl11:  dbra    d7,vbl10                ; loop for more vectors

*--- monitor screen dump flag
vbl12:  clr.l   a5                      ; quick zeropage
        tst.w   _prtcnt(a5)             ; printscreen active?
        bne     no_print                ; no

*+
* printScreen
*
* We re-enable vblanks here, until the printScreen finishes.
*
*-
        bsr     _scrdmp                 ; dump screen
        move.w  #-1,_prtcnt             ; unlock printScreen
no_print:


*--- restore registers & return (and a handy RTE)
        movem.l (sp)+,d0-d7/a0-a6
vblret: addq.w  #1,vblsem               ; V(vblsem) [release vblank]
_rte:   rte



*+
* wvbl - wait for next vblank
* Passed:       nothing
* Returns:      at beginning of next vblank
* Uses:         D0
*-
wvbl:
        move.w  sr,-(sp)                ; save psw
        and.w   #$ffff-$700,sr          ; enable vbl interrupts
        move.l  _frclock,d0             ; d0 = frame clock
wvbl1:  cmp.l   _frclock,d0             ; wait for clock to change
        beq     wvbl1
        move.w  (sp)+,sr                ; then restore psw & return
        rts



*+
* _critic - critical error handler binding for C
* Falls-into:   _critich
* (screwy way to save two bytes...)
*
*-
_critic:
        move.l  etv_critic,-(sp)        ; jump through critic vector
```

```
*+
* _critich - default critical error handler
* Loads -1 into D0 and returns.
*
*-
_critich:
        moveq   #-1,d0                  ; default return value = ERROR
        rts                             ; return to trap invoker


*+
* trp13h - GEMDOS BIOS trap handler (trap 13)
* trp14h - Atari BIOS extensions (trap 14)
* traph  - trap handler
*
* On the stack:
*       From super-             From user
*       visor mode:             mode:
*       -----------             ------------
*       N(sp) args              N(usp) args
*       6(sp) func#             6(usp) func#
*       2(sp) ret               2(ssp) ret
*        (sp) SR                 (ssp) SR
*
* Returns:      anything in D0
*
* Uses:         d0-d2/a0-a2
* Keeps:        C registers
*
* Notes:        BIOS traps are re-entrant to 'nlevels' (declared near the
*               beginning of this file).  Attempts to recurse more than
*               'nlevels' will probably result in a crash.
*
*               BIOS calls may be made from user mode.  (This differs from
*               the current GEMDOS spec, which states that BIOS traps are
*               available from supervisor mode only).
*
*-
trp14h: lea     trp14tab(pc),a0         ; a0 -> trap14 jump table
        bra.s   traph
trp13h: lea     trp13tab(pc),a0         ; a0 -> trap13 jump table

* save registers, twiddle stack:
traph:  move.l  savptr,a1               ; a1 -> register save area
        move.w  (sp)+,d0                ; pop SR and save it
        move.w  d0,-(a1)                ; (need in D0 for user-mode test)
        move.l  (sp)+,-(a1)             ; save return addr
        movem.l d3-d7/a3-a7,-(a1)       ; save C registers + super stack
        move.l  a1,savptr               ; update save-area pointer

* make sure we have the right stack, call function:
        btst    #13,d0                  ; was in user mode?
        bne     b_supr                  ; (was in super: use super stack)
        move.l  usp,a7                  ; use user stack
b_supr: move.w  (sp)+,d0                ; get function#
        cmp.w   (a0)+,d0                ; out of range?
        bge     b_exit                  ; (yes, so punt)
        lsl.w   #2,d0                   ; turn d0 into longword index
        move.l  (a0,d0.w),d0            ; get pointer to function handler
        move.l  d0,a0                   ; (quick and dirty test-for-negative)
        bpl     b_1                     ; points to code
        move.l  (a0),a0                 ; indirect through RAM...
b_1:    clr.l   a5                      ; a5 -> zero page
        jsr     (a0)                    ; call BIOS function

* restore registers, cleanup stack and return:
b_exit: move.l  savptr,a1               ; a1 -> register save area
        movem.l (a1)+,d3-d7/a3-a7       ; restore C registers + super stack
        move.l  (a1)+,-(sp)             ; push return address
        move.w  (a1)+,-(sp)             ; push old SR
        move.l  a1,savptr               ; update save-pointer
        rte                             ; return to caller


*------ jump table for GEMDOS functions:
trp13tab:
        dc.w    12                      ; number of entries in jump table
        dc.l    _get_mpb                ; 0: get memory parameter block
        dc.l    bconstat                ; 1: console status (input)
        dc.l    bconin                  ; 2: console input
        dc.l    bconout                 ; 3: console output
        dc.l    hdv_rw+$80000000        ; 4: [indirect] disk read/write
        dc.l    _setexc                 ; 5: set exception vector
        dc.l    _tickcal                ; 6: return tick calibration
        dc.l    hdv_bpb+$80000000       ; 7: [indirect] get BPB
        dc.l    bcostat                 ; 8: console status (output)
        dc.l    hdv_mediach+$80000000   ; 9: [indirect] media change inquiry
        dc.l    _drvmap                 ; 10: get active-drive bit vector
        dc.l    _shift                  ; 11: get/set keyboard shift bits



*------ jump table for Atari BIOS extensions:
trp14tab:
        dc.w    40                      ; number of entry points
        dc.l    initmous                ; 0: initialize mouse
        dc.l    _rts                    ; 1: (reserved)
        dc.l    _physbase               ; 2: get physical screen base
        dc.l    _logbase                ; 3: get logical screen base
        dc.l    _getrez                 ; 4: get screen resolution
        dc.l    _setscreen              ; 5: set video parameters
        dc.l    _setpalette             ; 6: set palette
        dc.l    _setcolor               ; 7: set single color
        dc.l    _floprd                 ; 8: read floppy sector(s)
        dc.l    _flopwr                 ; 9: write floppy sector
        dc.l    _flopfmt                ; 10: format floppy track
        dc.l    _getdsb                 ; 11: get device status block ptr

        dc.l    midiws                  ; 12: write string to MIDI port
        dc.l    mfpint                  ; 13: initialize MFP interrupt
        dc.l    iorec                   ; 14: set I/O record
        dc.l    rsconf                  ; 15: configure RS-233 communications
        dc.l    keytrans                ; 16: set keyboard translation tables

        dc.l    _rand                   ; 17: generate 24-bit random number
        dc.l    _proto_bt               ; 18: prototype boot sector
        dc.l    _flopver                ; 19: floppy verify

        dc.l    _dumpit                 ; 20: dump screen
        dc.l    _cursconf               ; 21: get/set cursor configuration
        dc.l    settime                 ; 22: set ikbd time
        dc.l    gettime                 ; 23: get ikbd time
        dc.l    bioskeys                ; 24: reset keyboard to powerup default
        dc.l    ikbdws                  ; 25: write string to ikbd

        dc.l    jdisint                 ; 26: disable mfp interrupt
        dc.l    jenabint                ; 27: enable mfp interrupt
        dc.l    giaccess                ; 28: read/write sound chip
        dc.l    offgibit                ; 29: reset bit in sound chip register
        dc.l    ongibit                 ; 30: set bit in sound chip register
        dc.l    xbtimer                 ; 31: initialize mfp timer
        dc.l    dosound                 ; 32: startup sound daemon
        dc.l    setprt                  ; 33: get/set printer configuration
        dc.l    ikbdvecs                ; 34: return ptr to base of kbd vars
        dc.l    kbrate                  ; 35: get/set keyboard repeat rate
        dc.l    _prtblk                 ; 36: _prtblk primitive
        dc.l    wvbl                    ; 37: wait for next vblank
        dc.l    supexec                 ; 38: execute in super mode
        dc.l    puntaes                 ; 39: throw away AES


*+
* supexec - execute some code in supervisor mode
*
*-
supexec:
        move.l  4(sp),a0                ; a0 -> code
        jmp     (a0)                    ; execute it


*+
* Character device I/O
*
* No check is made for "bogus" device numbers.  A wierd device
* number will result in a crash.
*
*-
bconstat: lea   tconstat(pc),a0         ; a0 -> stat table
        bra.s   chsw

bconin: lea     tconin(pc),a0           ; a0 -> input table
        bra.s   chsw

bcostat: lea    tcostat(pc),a0          ; a0 -> ostat table
        bra.s   chsw
```

```
bconout: lea     tconout(pc),a0                 ; a0 -> output table
chsw:    move.w  4(sp),d0                       ; get device number
         lsl.w   #2,d0                          ; turn into longword index
         move.l  (a0,d0.w),a0                   ; get address of handler
         jmp     (a0)                           ; jump to it


*+
* Jump tables for
*        0 - lst: (printer)
*        1 - aux: (rs232)
*        2 - con: (screen)
*        3 - Atari midi
*        4 - Atari keyboard (output only)
*        5 - raw console output (bypass vt52 pressure cooker)
*
* No range checking is performed.  If a bogus device number
* is passed to the BIOS' character I/O handler, the system
* will crash or become funky duex.
*
*-
tconstat: dc.l _rts,auxistat,constat,midstat,_rts,_rts
tconin:   dc.l _lstin,auxin,conin,midin,_rts,_rts
tcostat:  dc.l _lstostat,_auxostat,conoutst,ikbdost,midiost,_rts
tconout:  dc.l _lstout,_auxout,conout,midiwc,ikbdwc,_asc_out



*+
* _drvmap - return "active drive" bit vector
* Passed:          nothing
* Returns:         D0.L = a bit vector of live (rwabs'able) block devices
*
*-
_drvmap
        move.l   _drvbits(a5),d0
        rts

*+
* _shift - get/set keyboard shift state
* Synopsis:        LONG _shift(bits)
*                  WORD bits
*
* Returns:         D0.B = shift/alt/ctl/shift' bits
*
* Note:            Since the shift bits are changed at interrupt
*                  level, any set from a get of the shift state
*                  must be done as a critical section.
*
*-
_shift:
        moveq    #0,d0
        move.b   kbshift(a5),d0
        move.w   4(sp),d1
        bmi      shifr
        move.b   d1,kbshift(a5)
```

```
shifr:  rts


*+
* _get_mpb - return initial memory parameter block
* Synopsis:       _get_mpb(mpb)
*                 MPB *mpb;
*
* Returns:        The properly initialized MPB.
*                 The MPB points to an MD somewhere in BSS.  The MD /must/
*                 be in RAM since DOS will modify it.
*-
_get_mpb:
        move.l  4(sp),a0                        ; a0 -> MPB
        lea     themd(a5),a1                    ; a1 -> MD

*--- initialize MPB:
        move.l  a1,(a0)                         ; mp_mfl = &themd;
        clr.l   4(a0)                           ; mp_mal = NULL;
        move.l  a1,8(a0)                        ; mp_rover = &themd;

*--- initialize MD:
        clr.l   (a1)                            ; m_link = NULL;
        move.l  _membot(a5),4(a1)               ; m_start = _membot;
        move.l  _memtop(a5),d0                  ; m_length = _memtop - _membot;
        sub.l   _membot(a5),d0
        move.l  d0,8(a1)
        clr.l   $c(a1)                          ; m_own = NULL;
        rts


*+
* _setexc - set exception vector
* Synopsis:       setexc(vecno, addr)
*                 If 'addr' < 0, the vector is not set.
*
*                 Extended vectors ($100 through $107) are located in the
*                 first eight longwords of BSS, at $400.  This is for
*                 convienience -- they could really be located anywhere.
*
* Returns:        D0.L = original vector value
*
*-
_setexc:
        move.w  4(sp),d0                        ; d0 = vector#
        lsl.w   #2,d0                           ; turn into longword index
        clr.l   a0
        lea     (a0,d0.w),a0                    ; a0 -> vector
        move.l  (a0),d0                         ; d0 = current vector address
        move.l  6(sp),d1                        ; d1 = what_to_change_it_to
        bmi     setex1                          ; punt if (d1 < 0)
        move.l  d1,(a0)                         ; set vector address
setex1: rts


*+
* _tickcal - return system timer calibration value (in ms)
*
*-
_tickcal:
        clr.l   d0                      ; cast to unsigned longword
        move.w  _timr_ms(a5),d0         ; get calibration
        rts


*+
* _physbase - get physical display base
*
*-
_physbase:
        moveq   #0,d0                   ; cleanup pointer-to-be
        move.b  dbaseh,d0               ; load and shift bits 16..23
        lsl.w   #8,d0
        move.b  dbasel,d0               ; load and shift bits 8..15
        lsl.l   #8,d0
        rts                             ; return pointer in d0


*+
* _logbase - get logical display base
*
*-
_logbase:
        move.l  _v_bas_ad(a5),d0        ; set software shadow
        rts


*+
* _getrez - get current screen rez
*
*-
_getrez:
        moveq   #0,d0                   ; cleanup dirty bits
        move.b  shiftmd(a5),d0          ; get screen rezolution
        and.b   #$03,d0                 ; strip garbage bits
        rts                             ; return rez


*+
* _setscreen - set screen location(s), rez
*       _setscreen(logicalLoc, physicalLoc, rez)
*       LONG logicalLoc, physicalLoc;
*       WORD rez;
*
*-
_setscreen:

*--- set logical location:
        tst.l   4(sp)                   ; if(logloc < 0) then ignore it
        bmi     f5a
        move.l  4(sp),_v_bas_ad(a5)     ; set software pointer from logloc
```

```
*--- set physical location:
f5a:    tst.l   8(sp)                   ; if(physloc < 0) then ignore it
        bmi     f5b
        move.b  9(sp),dbaseh            ; set bits 16..23 of hardware pointer
        move.b  $a(sp),dbasel           ; set bits 8..15 of hardware pointer

*--- change screen resolution (clears the screen, clobbers the cursor):
f5b:    tst.w   $c(sp)                  ; if(rez < 0) then ignore it
        bmi     f5r
        move.b  $d(sp),sshiftmd(a5)     ; set software shadow
        bsr     wvbl                    ; wait for start of vertical-blank
        move.b  sshiftmd(a5),shiftmd    ; set hardware location
        clr.w   vblsem(a5)              ; disable vblank processing
        jsr     esc_init                ; re-initialize glass tty routines
        move.w  #1,vblsem               ; re-enable vblanks
f5r:    rts


*+
* _setpalette - set palette (on next vblank)
*       _setpallete(LONG palettePtr)
*
*-
_setpalette:
        move.l  4(sp),colorptr(a5)      ; set software pointer
        rts                             ; (updated by vbl handler)


*+
* _setcolor - set single color, return old color
*       _setcolor(WORD colorNum, WORD colorValue)
*
*-
_setcolor:
        move.w  4(sp),d1                ; get color number
        add.w   d1,d1                   ; turn into word index
        and.w   #$1f,d1                 ; force color range (prevent buserr)
        lea     color0,a0               ; a0 -> base of palette memory
        move.w  (a0,d1.w),d0            ; return old color
        and.w   #$0777,d0               ; mask dirty bits
        tst.w   6(sp)                   ; if new color is <0, don't set it
        bmi     _setc1                  ; (punt)
        move.w  6(sp),(a0,d1.w)         ; set new color
_setc1: rts


*+
* puntaes - throw-away AES, restart the system
*  Passed:      nothing
*  Uses:        everything
*  Returns:     if AES was already thrown away
*
*-
puntaes:
        move.l  os_magic(pc),a0         ; get pointer to magic
        cmp.l   #$87654321,(a0)         ; is the magic still there?
        bne     paes1                   ; no -- just return

        cmp.l   phystop,a0              ; is it in ROM?
        bge     paes1                   ; yes -- we can't do anything about it
        clr.l   (a0)                    ; clobber AES!
        bra     reseth                  ; restart the system

paes1:  rts


*+
* _term - terminate current process
* Called-by:    Uncaught traps (bus errors, and so on)
* Saves:        processor state (in a bailout area)
*
*-
_term:
        bsr     savp_2                  ; stack PC
        nop                             ; (never executed)
savp_2: move.l  (sp)+,proc_pc           ; save bogus PC + exception number
        movem.l d0-d7/a0-a7,proc_regs   ; common registers
        move.l  usp,a0                  ; save USP
        move.l  a0,proc_usp
        move.w  #15,d0                  ; save 16 words off top of
        lea     proc_stk,a0             ; the stack (enough for
        move.l  sp,a1                   ; any possible 68000 exception)
savp_1: move.w  (a1)+,(a0)+             ; save a word
        dbra    d0,savp_1
        move.l  #$12345678,proc_lives   ; set magic number (procdump lives)

*--- draw an appropriate number of 'shrooms on the screen:
        moveq   #0,d1
        move.b  proc_pc,d1
        subq    #1,d1                   ; 2 for bus error, 3 for address, etc.
        bsr     do_shroom

        move.l  #savend,savptr          ; clobber BIOS top level
        move.w  #1,-(sp)                ; "error" return condition
        clr.l   -(sp)                   ; GEMDOS function #0
        trap    #1                      ; "terminate process"
        bra     reseth                  ; on return, reset system


*+
* do_shroom - draw little mushroom clouds on the screen
*  Passed:      d1.w = #shrooms to draw (DBRA count)
*  Returns:     some shrooms on display
*  Uses:        d0-d7/a0-a2
*
*  Discussion:  The graphics ain't all that great.  And this is silly.
*
*-
do_shroom:
        move.b  shiftmd,d7
        and.w   #$0003,d7
        add.w   d7,d7                   ; d7 = rez index
        clr.l   d0
        move.b  dbaseh,d0
        lsl.w   #8,d0
        move.b  dbasel,d0
        lsl.l   #8,d0
        move.l  d0,a0
        add.w   mindex(pc,d7.w),a0      ; a0 -> base of mem to draw at

        lea     mushroom(pc),a1         ; a1 -> source form
        move.w  #15,d6                  ; d6 = scanline count

dm0:    move.w  d1,d2                   ; d3 = # to draw on this line
        move.l  a0,a2                   ; save ptr to beg of line
dm1:    move.w  mcount(pc,d7.w),d5      ; d5 = #words to replicate
dm2:    move.w  (a1),(a0)+              ; draw a word
        dbra    d5,dm2                  ; (complete single shroom)
        dbra    d2,dm1                  ; another, on the same line
        addq    #2,a1                   ; next source word
        add.w   mwidth(pc,d7.w),a2      ; next dest line
        move.l  a2,a0
        dbra    d6,dm0                  ; (loop for next line)
        rts                             ; byebye

mindex:         dc.w    100*160,100*160,200*80
mcount:         dc.w    3,1,0
mwidth:         dc.w    160,160,80

*--- what it is:
mushroom:
        dc.w    %0000011111000000
        dc.w    %0001111111110000
        dc.w    %0011101111111000
        dc.w    %0111011111110100
        dc.w    %1011011111111010
        dc.w    %1011101111111010
        dc.w    %1101111111110110
        dc.w    %0110011011111100
        dc.w    %0011001010001000
        dc.w    %0000001010000000
        dc.w    %0000010001000000
        dc.w    %0000010001000000
        dc.w    %0000010101000000
        dc.w    %0000010100100000
        dc.w    %0000100100100000
        dc.w    %0000100100100000
        dc.w    %0001001010010000


*+
* _fastcpy - "fast" 512-byte copy
* Synopsis:     fastcpy(src, dest)
*
*               Used by _rwabs to fake disk DMA to odd addresses.  Therefore,
*               disk I/O on odd addresses is very slow.  Lose, lose.
*
```

```
*-
_fastcpy:
        move.l  4(sp),a0                ; a0 -> src
        move.l  8(sp),a1                ; a1 -> dest
        move.w  #63,d0                  ; d0 = move count (64*8 = 512)
fast1:  move.b  (a0)+,(a1)+             ; copy 8 bytes at a time
        move.b  (a0)+,(a1)+             ;       to minimize loop overhead
        move.b  (a0)+,(a1)+
        move.b  (a0)+,(a1)+
        move.b  (a0)+,(a1)+
        move.b  (a0)+,(a1)+
        move.b  (a0)+,(a1)+
        move.b  (a0)+,(a1)+
        dbra    d0,fast1
        rts


*+
* Go through hard-disk initialization vector
*
*-
_hinit: move.l  hdv_init,-(sp)
        rts


autopath:       dc.b    '\AUTO\'
autofile:       dc.b    '*.PRG',0
                dc.w    $1234,$5678,$9abc,$def0
        even

*+
* _auto - exec auto-startup files in the appropriate subdirectory
* _auto1 - exec (with filename args)
* Passed:       a0 -> full filespec (pathname)
*               a1 -> filename part of filespec
*               _drvbits: bit vector of active drives
*               _bootdev: contains device to exec from
*
* Returns:      nothing
*
* Note:         If _drvbits%%_bootdev is zero, _auto simply quits (since
*               the device isn't active....)
*
* Uses:         everything
*-
        .globl  _auto                   ; for debugging
_auto:  lea     autopath(pc),a0         ; -> path
        lea     autofile(pc),a1         ; -> filename

_auto1: move.l  (sp)+,autoret           ; return addr (used by execlr)
        clr.l   a5                      ; quick zeropage
        move.l  a0,pathname(a5)         ; setup filename/pathname ptrs
        move.l  a1,filename(a5)

        move.l  _drvbits(a5),d0         ; d0 = active dev vector
        move.w  _bootdev,d1             ; d1 = dev# to exec from
        btst    d1,d0                   ; is the dev alive?
        beq     autoq                   ; (no -- so punt)

        lea     nullenv(pc),a0          ; a0 -> \0\0
        move.l  a0,-(sp)                ; null enviroment
        move.l  a0,-(sp)                ; null command tail
        move.l  a0,-(sp)                ; null shell name
        move.w  #5,-(sp)                ; Create-PSP subfunction
        move.w  #$4b,-(sp)              ; exec function#
        trap    #1                      ; do DOS call
        add.w   #16,sp

        move.l  d0,a0                   ; a0 -> PSP
        move.l  #fauto,8(a0)            ; initial PC -> autoexec prog

        move.l  a3,-(sp)                ; null enviroment
        move.l  d0,-(sp)                ; -> PSP
        move.l  a3,-(sp)                ; null shell name
        move.w  #4,-(sp)                ; just-go
        move.w  #$4b,-(sp)              ; function = exec
        trap    #1                      ; do it
        add.w   #16,sp                  ; cleanup stack & goodbye
autoq:  rts


*+
* fauto - exec'd by _auto to do autostartup
*
* Passed:       pathname -> path part of filespec
*               filename -> file part of filespec
*
*-
fauto:
        clr.l   -(sp)                   ; get into super mode
        move.w  #$20,-(sp)
        trap    #1
        addq    #6,sp                   ; cleanup
        move.l  d0,a4                   ; a4 -> saved super stack

*--- free up some memory:
        move.l  4(a7),a5                ; a5 -> base page
        lea     $100(a5),sp             ; sp -> new, safer addr
        move.l  #$100,-(sp)             ; keep $100 (just the basepage)
        move.l  a5,-(sp)                ; -> start of mem to keep
        clr.w   -(sp)                   ; junk word
        move.w  #$4a,-(sp)              ; setblock(...)
        trap    #1
        addq    #6,sp
        tst.w   d0
        bne     au_dn                   ; punt on error

        move.w  #$0007,-(sp)            ; find r/o+hidden+system files
        move.l  pathname,-(sp)          ; -> filename (on input)
        move.w  #$4e,-(sp)              ; searchFirst
        moveq   #8,d7                   ; d7 = cleanup amount
au1:    pea     autodma                 ; setup DTA (for search)
        move.w  #$1a,-(sp)
        trap    #1
        addq    #6,sp

        trap    #1                      ; search first/search next
        add.w   d7,sp                   ; cleanup stack
        tst.w   d0                      ; test for match
        bne     au_dn                   ; (no match -- quit)

*--- construct filename from path and the name we just found:
        move.l  pathname,a0             ; copy pathname
        move.l  filename,a2             ; a2 -> end+1 of pathname
        lea     autoname,a1
au3:    move.b  (a0)+,(a1)+             ; copy path part of name
        cmp.l   a0,a2                   ; finished?
        bne     au3                     ; (no)
        lea     autodma+30,a0           ; copy fname to end of pathname
au2:    move.b  (a0)+,(a1)+
        bne     au2

        pea     nullenv(pc)             ; null enviroment
        pea     nullenv(pc)             ; no command tail
        pea     autoname                ; -> file to exec
        clr.w   -(sp)                   ; load-and-go
        move.w  #$4b,-(sp)              ; exec(...)
        trap    #1
        add.w   #16,sp

        moveq   #2,d7                   ; reset cleanup amount
        move.w  #$4f,-(sp)              ; searchNext
        bra     au1

*+
* The first GEMDOS process can never terminate.
* This is not a good feature.
* Kludge around it -- re-initialize the stack
* and return to the guy who called us to begin with.
*
*-
au_dn:  lea     _supstk+2048,sp         ; setup supervisor stack
        move.l  autoret,-(sp)           ; get return addr
        rts                             ; just jump there ...


*--- bss for auto-exec:
        bss
autoret:        ds.l    1               ; -> _auto's caller (yeccch)
pathname:       ds.l    1               ; -> filespec's pathname
filename:       ds.l    1               ; -> filename part of path
autodma:        ds.b    44              ; 44 bytes for directory search
autoname:       ds.b    32              ; 32 bytes for path+filename
        even

        text
```

```
*+
* _dumpit: dump screen
*
*-
_dumpit:
        clr.w   _prtcnt
        bsr     _scrdmp
        move.w  #$ffff,_prtcnt
        rts


*+
* _scrdmp - printScreen(), front-end to _prtblk()
*  Passed:         nothing
*  Returns:        nothing
*  Uses:           everything
*
*-
_scrdmp:
        clr.l   a5                              ; easy zeropage
        move.l  _v_bas_ad(a5),p_blkptr(a5)      ; -> screen mem
        clr.w   p_offset(a5)                    ; offset = 0
        clr.w   d0
        move.b  sshiftmd(a5),d0                 ; get w & h
        move.w  d0,p_srcres(a5)
        add.w   d0,d0
        lea     reztab(pc),a0
        move.w  (a0,d0.w),p_width(a5)           ; set display width, height
        move.w  6(a0,d0.w),p_height(a5)
        clr.w   p_left(a5)                      ; left = right = 0
        clr.w   p_right(a5)
        move.l  #$ff8240,p_colpal(a5)           ; -> hardware palettes
        clr.w   p_masks(a5)                     ; default masks ptr

* draft or final mode
        move.w  pconfig(a5),d1                  ; p_dstres = pconfig%%3
        lsr.w   #3,d1
        and.w   #1,d1
        move.w  d1,p_dstres(a5)

* printer or rs232 port
        move.w  pconfig(a5),d1                  ; p_port = pconfig%%4
        move.w  d1,d0
        lsr.w   #4,d0
        and.w   #1,d0
        move.w  d0,p_port(a5)

* select printer flavor
        and.w   #7,d1                           ; p_type = ptype[pconfig & 7]
        move.b  ptype(pc,d1.w),d0
        move.w  d0,p_type

* do it
        pea     prtargs(a5)                     ; -> beginning of parameter area
        bsr     _prtblk                         ; print it (finally)
        addq    #4,sp                           ; cleanup stack
        rts                                     ; and return


*--- screen resolution table (pixels) for printScreen
reztab: dc.w    320,640,640                     ; widths
        dc.w    200,200,400                     ; heights


*--- printer flavors (based on low 3 bits of pconfig)
ptype:
        dc.b    0                               ; atari mono dot
        dc.b    2                               ; atari mono daisy
        dc.b    1                               ; atari color dot
        dc.b    -1                              ; [atari color daisy???]
        dc.b    3                               ; epson mono dot
        dc.b    -1                              ; [epson mono daisy]
        dc.b    -1                              ; [epson color dot]
        dc.b    -1                              ; [epson color daisy]
        even

*--- parameter storage for printScreen:
        bss
prtargs:
p_blkptr:       ds.l    1                       ; -> bitmap to print
p_offset:       ds.w    1                       ; offset on page
p_width:        ds.w    1                       ; width and height
p_height:       ds.w    1
p_left:         ds.w    1                       ; left & right leading
p_right:        ds.w    1
p_srcres:       ds.w    1                       ; source rez (0, 1, 2)
p_dstres:       ds.w    1                       ; destination rez (0, 1)
p_colpal:       ds.l    1                       ; -> color palettes
p_type:         ds.w    1                       ; printer type (0, 1)
p_port:         ds.w    1                       ; printer port (0, 1)
p_masks:        ds.l    1                       ; -> halftone masks
```

```
*-----------------------------------------------------------------------
*                                                                      :
*       Position-independent OS mover                                  :
*       (C)1985 Atari Corp.                                            :
*                                                                      :
*       Takes over from the Loader,                                    :
*       cleans up the display;                                         :
*       moves RAM-loaded OS from where it is to where it should be.    :
*                                                                      :
* 23-May-1985 lmd       Re-write from old, crufty version.             :
*                                                                      :
*-----------------------------------------------------------------------


*--- interface equates to OS:
lowstart        equ     $580            ; start of low BSS to clear
src_offset      equ     $100            ; offset from 'start' to OS image
os_size         equ     $38000          ; size of OS


*--- hardware:
dbaselo         equ     $ff8203         ; display base low (really, medium)
dbasehi         equ     $ff8201         ; display base high
color0          equ     $ff8240         ; base of palette mem
gpip            equ     $fffa01         ; general porpoise input



*+
* Take control from the Loader;
* turn on interrupts and clean up the screen:
*
*-
start:  move.w  #$2700,sr               ; supermode, no interrupts
        bsr     ramp                    ; cleanup display

        lea     start(pc),a0            ; a0 -> base of loaded OS
        lea     src_offset(a0),a0
        move.l  8(a0),a1                ; a1 = a2 = a3 -> destination
        move.l  a1,a2                   ; a2 -> saddr
        move.l  a1,a3                   ; a3 -> dest
        move.w  #(os_size/16)-1,d0      ; d0 = d1 = size (16-byte chunks)
        move.w  d0,d1

*--- copy OS to destination:
mvit:   move.l  (a0)+,(a1)+             ; copy 16 bytes /fast/
        move.l  (a0)+,(a1)+
        move.l  (a0)+,(a1)+
        move.l  (a0)+,(a1)+
        dbra    d0,mvit                 ; ...until we're done

*--- startup the system:
        jmp     (a2)                    ; jump to OS base addr
```

```
*+
* ramp - pretty transition from boot screen (the Fog)
* Takes about 0.5 seconds for a color display;
* No time atoll for a mono system.
*
*-
ramp:
        btst.b  #7,gpip                         ; are we mono?
        beq     itsmono                         ; yes, we ARE devo

*+
* a color monitor is attached (attatched)?
* anyway, bring up the fog....
*
*-
ramp_1: clr.l   d0                              ; assume we're done
        lea     color0,a0                       ; a0 -> palette RAM
        move.w  #15,d7                          ; d7 = count (do all colors)
ramp_2: move.w  (a0),d1                         ; get palette bits
        and.w   #$777,d1                        ; strip garbage ones
        cmp.w   #$777,d1                        ; are we already at white?
        beq     ramp_3                          ; (yes, so don't increment this one)

*--- bump color up one notch:
        move.w  #$700,d2                        ; d2 = mask
        moveq   #2,d3                           ; d3 = count (do this three times)
ramp_4: move.w  d1,d4                           ; d4 = color & mask
        and.w   d2,d4
        move.w  #$777,d5                        ; d5 = $777 & mask
        and.w   d2,d5
        cmp.w   d5,d4                           ; if we're already at 7, just continue
        beq     incq
        move.w  d2,d4                           ; d4 = $111 & mask
        and.w   #$111,d4
        add.w   d4,d1                           ; d1 += d4; bump the color
        moveq   #1,d0                           ; not done yet (set notDone flag)
incq:   lsr.w   #4,d2                           ; shift the mask down four bits
        dbra    d3,ramp_4                       ; do some more fields

ramp_3: move.w  d1,(a0)+                        ; shove new value into palette register
        dbra    d7,ramp_2                       ; loop for more registers

        move.w  #$6000,d1                       ; delay a while
ramp_d: dbra    d1,ramp_d

        tst.l   d0                              ; are all palettes at $x777?
        bne     ramp_1                          ; (no -- so ramp again)


*+
* Done with the ramp
* (or, we're on a mono system).
*
```

```
* Clobber last 32K of a 512K system
* and move the display there.
*
*-
itsmono:
        lea     $78000,a0               ; a0 -> base of new display
        move.w  #$7ff,d0
        moveq   #0,d1                   ; cheap zero
zap:    move.l  d1,(a0)+                ; clear 16 bytes /real fast/
        move.l  d1,(a0)+
        move.l  d1,(a0)+
        move.l  d1,(a0)+
        dbra    d0,zap                  ; ...$800 times....

        move.b  #$07,dbasehi            ; point display at new base
        move.b  #$80,dbaselo
        rts
```

```
/*
 * Initialize OS
 * Start something up (either GEM or COMMAND.COM).
 * Return when that thing is done.
 *
 *  4-Mar-1985 lmd      Cleanup (removed some lint)
 *  4-Mar-1985 lmd      Wired for new GEM system.
 * 11-Mar-1985 lmd      split out osi()
 * 13-Mar-1985 lmd      migrated bufl[] to base BSS (for future expansion)
 * 13-Mar-1985 lmd      ripped out main() [why keep this file around?]
 */


#include "fs.h"

extern long oscall();
#define xexec(a,b,c,d) oscall(0x4b,a,b,c,d)

/*
 * Sector buffers,
 * four seems to be about right (hard-coded in osi())
 * Extensible through base-BSS links.
 */
char secbuf[4][512];                            /* sector buffers */
BCB bcbx[4];                                    /* bcb for each buffer */



/*
 * Initialize GEMDOS
 */
osi()
{
        extern BCB *bufl[2];                    /* two buffer lists */
        extern int bootdev;
        extern int cmdload;

        /*
         * Setup sector buffers (four of 'em)
         */
        bcbx[0].b_link = &bcbx[1];
        bcbx[2].b_link = &bcbx[3];
        bcbx[0].b_bufdrv = -1;
        bcbx[1].b_bufdrv = -1;
        bcbx[2].b_bufdrv = -1;
        bcbx[3].b_bufdrv = -1;
        bcbx[0].b_bufr = &secbuf[0][0];
        bcbx[1].b_bufr = &secbuf[1][0];
        bcbx[2].b_bufr = &secbuf[2][0];
        bcbx[3].b_bufr = &secbuf[3][0];

        /*
         * Setup links in buffer-list
         * First one caches FATs,
         * second one caches directory and data blocks.
         */
	bufl[0] = &bcbx[0];			/* fat buffers */
	bufl[1] = &bcbx[2];			/* dir/data buffers */

	/*
	 * Initialize OS, login boot device:
	 */
	osinit();				/* initialize OS */
	xsetdrv(bootdev);			/* set default drive# */
}
```

```
#include "portab.h"
/* #define      DAS_BOOT 1
 */

/*
 * ST Disk support (and random BIOS functions)
 * (C)1985 Atari Corp.
 *
 *----
 * 23-Feb-1985 lmd      Added multiple-sector floppy read support.
 * 23-Feb-1985 lmd      Added "rand()" function.
 * 24-Feb-1985 lmd      Added hard disk hooks.
 * 24-Feb-1985 lmd      Added floppy and hard boot code.
 * 25-Feb-1985 lmd      boot() goes to default boot device
 * 28-Feb-1985 lmd      boot() returns diagnostics, initializes disk system
 *  1-Mar-1985 lmd      Added proto_bt() boot sector prototyper
 *  1-Mar-1985 lmd      Added mediach(dev) BIOS call
 *  4-Mar-1985 lmd      getbpb() sets disk mode to "SAFE"
 *  4-Mar-1985 lmd      fixed bugs in proto_bt()
 *  9-Mar-1985 lmd      Added critical error handler hook
 * 13-Mar-1985 lmd      getbpb() returns NULL on read failure
 * 17-Mar-1985 lmd      Added write-verify switch
 * 22-Mar-1985 lmd      Added magic r/w mode to rwabs (rw = 2, 3)
 *  1-Apr-1985 lmd      Moved DSBs to flop.s (hooray!)
 *  8-Apr-1985 lmd      Cleaned up installable dev interface
 * 15-Apr-1985 lmd      Happy IRS day.
 * 15-Apr-1985 lmd      check for dev>=2 (only floppies allowed...)
 *  6-May-1985 lmd      Added access-timing depended UNSURE checking
 *----
 */
#define MAXACCTIM       300L            /* 1.5seconds "free" time */

#define READ    0
#define WRITE   1

#define low8bits(x) ((x)&0xff)          /* unsigned coercion of char to int */



/*
 * Information we need from an IBM-PC-format
 * boot sector:
 */
#define VOL_SERIAL      0x08    /* (.A) 24-bit volume serial#   */
#define IBM_BPS         0x0b    /* (.W) #bytes/sector           */
#define IBM_SPC         0x0d    /* (.B) #sectors/cluster        */
#define IBM_RES         0x0e    /* (.W) #reserved sectors       */
#define IBM_NFATS       0x10    /* (.B) #FATs                   */
#define IBM_NDIRS       0x11    /* (.W) #root directory entries */
#define IBM_NSECTS      0x13    /* (.W) #sectors on media       */
#define IBM_MEDIA       0x15    /* (.B) media descriptor byte   */
#define IBM_SPF         0x16    /* (.W) #sectors/FAT            */
#define IBM_SPT         0x18    /* (.W) #sectors/track          */
#define IBM_NSIDES      0x1a    /* (.W) #sides on dev           */
#define IBM_NHID        0x1c    /* (.W) #hidden sectors         */
```

```
#define CRITICAL_RETRY  0x00010000L                     /* "retry" return code */

/*
 * Error codes
 * Sort of like the PC-DOS ones
 */
#define OK                              0               /* the anti-error */
#define ERROR                           (-1)            /* anti-success */
#define DRIVE_NOT_READY                 (-2)
#define UNKNOWN_CMD                     (-3)
#define CRC_ERROR                       (-4)
#define BAD_REQUEST                     (-5)
#define SEEK_ERROR                      (-6)
#define UNKNOWN_MEDIA                   (-7)
#define SECTOR_NOT_FOUND                (-8)
#define NO_PAPER                        (-9)            /* how can a disk do this? */
#define WRITE_FAULT                     (-10)
#define READ_FAULT                      (-11)
#define GENERAL_MISHAP                  (-12)           /* Captain_Catastrophe? */
#define WRITE_PROTECT                   (-13)
#define MEDIA_CHANGE                    (-14)
#define UNKNOWN_DEVICE                  (-15)
#define BAD_SECTORS                     (-16)           /* bad sectors on media */
#define INSERT_DISK                     (-17)           /* fake two drives */
#define WRONG_DISK_DUMMY                (-18)           /* luser stuck in wrong disk */


/*
 * BPB structure
 * as defined by GEMDOS:
 */
struct bpb {
        WORD    recsiz,                 /* physical sector size in bytes */
                clsiz,                  /* cluster size in sectors */
                clsizb,                 /* cluster size in bytes */
                rdlen,                  /* root directory length in sectors */
                fsiz,                   /* FAT size in sectors */
                fatrec,                 /* sector# of 1st sector of 2nd FAT */
                datrec,                 /* sector# of 1st data sector */
                numcl,                  /* number of data clusters on disk */
                bflags;                 /* various flags */
};


/*
 * Flags in bpb.bflags:
 */
#define BPB_16BIT_FAT   0x0001          /* indicates 16-bit FAT entries */


/*
 * "Device State Block"
 * as defined by us.
 * The DSB is used by drivers to hold a device's state.
 * Most devices require a pointer to this beastie as a parameter
 * in their calls.
 */
struct dsb {
        /*
         * Loaded (or computed from) the boot sector:
         */
        struct bpb b;                   /* JDOS' BPB */
        WORD    dntracks,               /* #tracks (cylinders) on dev */
                dnsides,                /* #sides per cylinder */
                dspc,                   /* #sectors/cylinder */
                dspt,                   /* #sectors/track */
                dhidden;                /* #hidden tracks */
        char    dserial[3];             /* 24-bit volume serial number */
} dsbtab[2];


/*
 * Variables maintained by floppy vblank monitor:
 */
extern char wpstatus[];         /* write-protect status */
extern char wplatch[];          /* write-protect status latch */
extern WORD motoron;            /* motor-on status (for both drives) */


/*
 * Other floppy variables:
 */
unsigned extern long hz_200;    /* system timer tick */
extern char diskbuf[];          /* disk buffer somewhere in BSS */
extern int nflops;              /* number of active floppies {0,1,2} */
unsigned extern long acctim[];  /* time of last floppy access */
long maxacctim;                 /* delay for floppy to turn UNSAFE */

char diskmode[2];               /* floppy mode {SAFE, UNSURE, CHANGED} */
int flopok[2];                  /* 0: drive OK; -1: drive unusable */
int curflop;                    /* current floppy# inserted */


/*
 * Floppy modes
 * (states for disk-change detection)
 */
#define SAFE    0               /* media has definitely not changed */
#define UNSURE  1               /* media might have changed (we don't know) */
#define CHANGED 2               /* media has definitely changed */



/*
 * dskinit - initialize floppy drives
 */
dskinit()
{
        LONG getbpb();
        extern LONG drvbits;

        WORD i, j;
        char *s, *d;

        maxacctim = MAXACCTIM;
        for (i = curflop = nflops = 0; i < 2; ++i)
        {
            diskmode[i] = SAFE;
            if ((flopok[i] = flopini(OL, OL, i, 0, 0, 0)) == 0)
            {
                ++nflops;
                drvbits |= 3;
            }
        }
}


/*
 * getdsb - return pointer to DSB
 */
LONG getdsb(dev)
WORD dev;
{
    return OL;
}


/*
 * getbpb - return pointer to BPB
 * Reset disk mode to "SAFE"
 */
long getbpb(dev)
WORD dev;
{
    register struct dsb *q;
    register struct bpb *p;
    register int i, j;
    char *s, *d;
    LONG ret, floprd(), critic();

    if (dev >= 2)                           /* only floppies here */
        return NULL;                        /* can't do much ... */

    q = &dsbtab[dev];                       /* pointer to DSB */
    p = &q->b;                              /* pointer to BPB */


    /*
     * Read the boot sector.
     * Compute the DOS BPB from the MSDOS one.
     */
    do {
        ret = floprd(diskbuf, OL, dev, 1, 0, 0, 1);
        if (ret < 0) ret = critic((WORD)ret, dev);
    } while (ret == CRITICAL_RETRY);
    if (ret < 0) return NULL;


    /*
     * If recsiz or clsiz turns out to be zero,
     * don't attempt to use the BPB.
     */
    if (!(i = u2i(diskbuf + IBM_BPS)) ||
        !(j = low8bits(diskbuf[IBM_SPC])))
            return NULL;


    /*
     * Build the BPB from the MSDOS-format information:
     */
    p->recsiz = i;
    p->clsiz = j;
    p->fsiz = u2i(diskbuf + IBM_SPF);
    p->fatrec = p->fsiz + 1;
    p->clsizb = p->recsiz * p->clsiz;
    p->rdlen = (u2i(diskbuf + IBM_NDIRS) << 5) / p->recsiz;
    p->datrec = p->fatrec + p->rdlen + p->fsiz;
    p->numcl = (u2i(diskbuf + IBM_NSECTS) - p->datrec) / p->clsiz;

    q->dnsides = u2i(diskbuf + IBM_NSIDES);               /* "extra" info */
    q->dspt = u2i(diskbuf + IBM_SPT);
    q->dspc = q->dnsides * q->dspt;
    q->dhidden = u2i(diskbuf + IBM_NHID);
    q->dntracks = u2i(diskbuf + IBM_NSECTS) / q->dspc;

    for (i = 0; i < 3; ++i)                               /* copy serial# */
        q->dserial[i] = diskbuf[VOL_SERIAL + i];

                                                          /* make safe/unsure */
    diskmode[dev] = (wplatch[dev] = wpstatus[dev]) ? UNSURE : SAFE;

    return (long)q;                                       /* return BPB ptr */
}


/*
 * mediach - determine if media has changed
 * Return SAFE if the media definitely has not changed.
 * Return UNSURE if we're not sure if it's changed.
 * Return CHANGED if we're sure the media changed.
 *
 */
WORD mediach(dev)
WORD dev;
{
    register WORD dv;
    register char *dm;

    if (dev >= 2)                        /* only floppies here */
        return UNKNOWN_DEVICE;
    dv = dev;
    dm = &diskmode[dv];

    if (*dm == CHANGED) return CHANGED;         /* always hack CHANGED  */
    if (wplatch[dv]) *dm = UNSURE;              /* ==> UNSURE           */
    if ((hz_200 - acctim[dv]) < maxacctim)      /* SAFE if within time limit */
        return SAFE;
    return *dm;                                 /* return UNSURE or SAFE */
}


/*
 * rwabs - read multiple sectors from dev, into a buffer:
 */
LONG rwabs(rw, buf, count, recno, dev)
WORD rw;
LONG buf;
WORD count, recno, dev;
{
    register int i;
    register WORD dv;
    register LONG rtn;
    register struct dsb *p;
    LONG ret;
    WORD mediach();
    LONG floprw();

    if (dev >= 2)                       /* only floppies here */
        return UNKNOWN_DEVICE;

    dv = dev;

    if (rw < 2)
    {
        p = &dsbtab[dv];

        /*
         * Check for media change.
         * If the media is UNSAFE, then read the boot sector to
         * determine if the media really was changed.
         * If the media was changed, return an error to the caller.
         */
        i = mediach(dv);
        if (i == CHANGED) return MEDIA_CHANGE;
        else if (i == UNSURE)
        {
            /*
             * Read boot sector and compare volume's serial number with
             * the one in the DSB.
             */
            do {
                ret = floprd(diskbuf, 0L, dv, 1, 0, 0, 1);
                if (ret < 0) ret = critic((WORD)ret, dv);
            } while (ret == CRITICAL_RETRY);
            if (ret < 0) return ret;
            for (i = 0; i < 3; ++i)
                if (diskbuf[VOL_SERIAL + i] != p->dserial[i])
                        return MEDIA_CHANGE;

            /* Reset write-protect latch */
            if (!(wplatch[dv] = wpstatus[dv]))
                diskmode[dv] = SAFE;
        }
    }

    if (!nflops) return DRIVE_NOT_READY;
    if (rw > 1) rw -= 2;                                /* fix magic r/w */
    return floprw(rw, buf, recno, dv, count);
}


/*
 * floprw - floppy read/write sectors
 */
LONG floprw(rw, buf, recno, dev, count)
WORD rw;
LONG buf;
WORD recno, dev, count;
{
    LONG critic(), flopver(), floprd(), flopwr();
    int u2i();
    extern WORD fverify;

    register struct dsb *p;
    register LONG ret;
    register WORD track, side, sect, cnt;
    WORD oddflag;
    LONG bf;


    p = &dsbtab[dev];
    oddflag = ((buf & 1) == 1);
    if (!p->dspc)                               /* "cannot happen" */
        p->dspt = p->dspc = 9;

    /*
     * Read or write sectors.
     * Optimize for multi-sector transfers
     * (as much of a track as possible):
     */
    while (count)
    {
        bf = oddflag ? diskbuf : buf;           /* choose a buffer */
        track = recno / p->dspc;                /* compute track# */
        sect = recno % p->dspc;                 /* compute sector# */
        if (sect < p->dspt)
            side = 0;                           /* single-sided media */
        else
        {                                       /* two-sided media */
            side = 1;
            sect -= p->dspt;
        }
        if (oddflag) cnt = 1;                           /* unaligned: read 1 sector */
        else if ((p->dspt - sect) < count)
             cnt = p->dspt - sect;                      /* rest of track */
        else cnt = count;                               /* part of track */

        ++sect;                                         /* physical sector number */

        do {
            if (rw)                                             /* write */
            {
                if (bf != buf) fastcpy(buf, bf);
                ret = flopwr(bf, 0L, dev, sect, track, side, cnt);

                if (!ret && fverify)                            /* verify */
                {
                    ret = flopver(diskbuf, 0L,
                                  dev, sect, track, side, cnt);
                    if (!ret && u2i(diskbuf))
                        ret = BAD_SECTORS;
                }
            }
            else                                                /* read */
            {
                ret = floprd(bf, 0L, dev, sect, track, side, cnt);
                if (bf != buf) fastcpy(bf, buf);
            }

            if (ret < 0)
                ret = critic((WORD)ret, dev);
        } while (ret == CRITICAL_RETRY);
        if (ret < 0) return ret;

        buf += ((long)cnt << 9);                        /* advance DMA pointer */
        recno += cnt;                                   /* bump record number */
        count -= cnt;                                   /* decrement count */
    }

    return OK;                                          /* success! */
}


#ifndef DAS_BOOT
/*
 * Random number generator parameters.
 * (from Knuth, vol II)
 */
#define RAND_A  3141592621L                     /* multiplier */
#define RAND_C  1                               /* incrementer */

LONG seed;                                      /* seed (zeroed at powerup) */

/*
 * Return a 24-bit random number.
 * If the seed is zero (uninitialized)
 * then use the frame clock, slightly
 * munged, as a starting value.
 */
LONG rand()
{
    extern LONG hz_200;                 /* raw 200-hz system timer counter */

    if (!seed) seed = hz_200 | (hz_200 << 16);
    seed = (RAND_A * seed + RAND_C);
    return (seed >> 8) & 0xffffff;
}
#endif


#define BOOT_MAGIC      0x1234          /* magic boot-sector checksum */


/*
 * Error returns:
 */
#define NO_DRIVE        1               /* no floppy attatched */
#define COULDNT_LOAD    2               /* couldn't read boot sector */
#define UNREADABLE      3               /* unreadable boot sector */
#define NOT_VALID_BS    4               /* boot sector not executable */


/*
 * Boot from floppy or hard disk.
 * Returns OK if diskbuf[] contains an executable
 * boot sector.
 */
boot()
{
    extern WORD _hinit();
    extern WORD bootdev;
    extern LONG floprd();
    register WORD err;


    /*
     * Initialize disk system:
     */
    hinit();

    /*
     * Attempt to load boot sector from floppy "bootdev":
     */
    err = nflops ? NO_DRIVE : COULDNT_LOAD;
    if (nflops && (bootdev < 2))
    {
        if (!floprd(diskbuf, 0L, bootdev, 1, 0, 0, 1))
            err = OK;
            else if (!wpstatus[0]) return UNREADABLE;
    }
    if (err != OK) return err;

    /*
```

```
	 * Successfully loaded boot sector from somewhere,
	 * check it out:
	 */
	return (checksum(diskbuf, 0x100) == BOOT_MAGIC) ? OK : NOT_VALID_BS;
}




#ifndef DAS_BOOT
/*
 * Prototype BPBs for floppies;
 * used to construct boot sectors.
 */
char proto_tab[] =
{			/* 40 tracks single sided */
	0x00,0x02,0x01,0x01,0x00,0x02,0x40,0x00,0x68,0x01,
	0xfc,0x02,0x00,0x09,0x00,0x01,0x00,0x00,0x00,

			/* 40 tracks double sided */
	0x00,0x02,0x02,0x01,0x00,0x02,0x70,0x00,0xd0,0x02,
	0xfd,0x02,0x00,0x09,0x00,0x02,0x00,0x00,0x00,

			/* 80 tracks single sided */
	0x00,0x02,0x02,0x01,0x00,0x02,0x70,0x00,0xd0,0x02,
	0xf8,0x05,0x00,0x09,0x00,0x01,0x00,0x00,0x00,

			/* 80 tracks double sided */
	0x00,0x02,0x02,0x01,0x00,0x02,0x70,0x00,0xa0,0x05,
	0xf9,0x05,0x00,0x09,0x00,0x02,0x00,0x00,0x00
};



/*
 * Prototype a boot sector.  (this is a strange function...)
 *
 * 'serial' is the disk's volume ID (or -1 not to initialize).
 * If serial > 0xffffff, it is replaced by a different, random serial number
 *
 * 'dsktyp' is the disk size (0, 1, 2, 3), or -1 not to initialize.
 *
 * If 'execflg' is 1, the boot sector is made executable (bootable);
 * If 'execflg' is 0, the boot sector is g'teed NOT to be executable;
 * If 'execflg' is -1, keep the boot sector the way it was passed
 * (it will stay executable or non-executable, no matter what other
 * changes were made to it).
 */
WORD proto_bt(buf, serial, dsksiz, execflg)
char *buf;
LONG serial;
WORD dsksiz, execflg;
{
	long rand();
	register int i, j;
```

```
    register char *s;
    WORD *p, w;


    /*
     * If execflg < 0, determine if boot sector is already executable.
     * Whatever the case, make sure the sector /stays/ the way it
     * came to us.
     */
    if (execflg < 0)
        execflg = (checksum(buf, 0x100) == BOOT_MAGIC);


    /*
     * Install volume ID
     */
    if (serial >= 0)
    {
        if (serial > 0x00ffffff)
            serial = rand();
        for (i = 0; i < 3; ++i)
        {
            buf[VOL_SERIAL + i] = serial & 0xff;
            serial >>= 8;
        }
    }


    /*
     * Install BPB
     */
    if (dsksiz >= 0)
    {
        j = dsksiz * 19;
        for (i = 0; i < 19; ++i)
            buf[IBM_BPS + i] = proto_tab[j++];
    }


    /*
     * Make the sector executable or non-executable.
     */
    w = 0;
    for (p = buf; p < (buf + 0x1fe);)
        w += *p++;
    *p = BOOT_MAGIC - w;
    if (!execflg) ++(*p);
}
#endif


/*
 * Compute checksum of a number of 16-bit words.
 */
WORD checksum(p, cnt)
WORD *p;
```

```
int cnt;
{
    register WORD i;

    for (i = 0; cnt--; )
        i += *p++;
    return i;
}



/*
 * Convert an 8086-flavored integer
 * to a 68000 integer.
 */
int u2i(loc)
char *loc;
{
    return (low8bits(*(loc+1)) << 8) | low8bits(*loc);
}
```

```
das_boot        equ     0
*-----------------------------------------------------------------------
*                                                                      :
*       130-ST / 520-ST                                                :
*       Floppy Disk Driver                                             :
*       (C)1985 Atari Corp.                                            :
*                                                                      :
* 22-Feb-1985 lmd       Added write-protect and motor-on monitoring.   :
* 22-Feb-1985 lmd       Substituted format-track for format-disk.      :
* 23-Feb-1985 lmd       Multiple-sector DMA in _floprd.                :
* 25-Feb-1985 lmd       _flopwr understands "ccount" (but cannot do    :
*                       multi-sector DMA -- a hardware constraint).    :
* 25-Feb-1985 lmd       Added "virgin" parameter to _flopfmt           :
* 27-Feb-1985 lmd       _flopwr() can write an entire track in one     :
*                       revolution of the disk....                     :
* 28-Feb-1985 lmd       _floprd() doesn't do reseek on seek error      :
*                       (it takes too long)                            :
*  4-Mar-1985 lmd       Added "bad sector" return to _flopfmt          :
*  7-Mar-1985 lmd       Fixed bug in _flopfmt bad sector return        :
*  8-Mar-1985 lmd       Fixed "floplock" and "flopulok" to save and    :
*                       restore C registers.                           :
* 10-Mar-1985 lmd       Added "disk flip" code (hook to _critic)       :
* 13-Mar-1985 lmd       If single-floppy system, copy drive 0's write- :
*                       protect transitions to drive 1.                :
* 13-Mar-1985 lmd       Set _wplatch after disk flip                   :
* 13-Mar-1985 lmd       Return reasonable error numbers                :
* 17-Mar-1985 lmd       Added _flopver()                               :
* 21-Mar-1985 lmd       dasBoot assembly switches, default seek rate   :
* 22-Mar-1985 lmd       format_track sets media change mode to CHANGED :
* 22-Mar-1985 lmd       a write to the boot sector sets the media      :
*                       change mode to UNSURE.                         :
* 28-Mar-1985 lmd       Force write-protect to "real time" mode        :
*                       on any exit from the driver.                   :
*  1-Apr-1985 lmd       Moved floppy DSBs to here.                     :
*  1-Apr-1985 lmd       Based variables off of zero-page               :
*  8-Apr-1985 lmd       Moved flock out of here to public basepage     :
* 30-Apr-1985 lmd       Disk errors set media-change mode to UNSURE    :
*  1-May-1985 lmd       Bug in _flopini; mis-use of args on stack      :
*  6-May-1985 lmd       Set _motoron nonzero on any floppy command.    :
*                       Added _acctim[] timer variables.               :
*                                                                      :
*-----------------------------------------------------------------------
        text


*------ Tunable values (subject to tweaking):
retries         equ     2               ; default # of retries - 1
midretry        equ     1               ; "middle" retry (when to reseek)
timeout         equ     $40000          ; short timeout (motor already on)
ltimeout        equu    $60000          ; long timeout (to startup motor)


*------ Exports:
        .globl _flopini                 ; init floppy                   func
        .globl _floprd                  ; read sector                   func
        .globl _flopvbl                 ; vertical blank monitor        func
```

```
 ifeq das_boot
        .globl _flopwr                        ; write sector                  func
        .globl _flopfmt                       ; format drive/track            func
        .globl _flopver                       ; verify sectors                func
 endc

        .globl  _wpstatus                     ; write-protect state (2 drives)
        .globl  _wplatch                      ; write-protect latch (2 drives)
        .globl  _motoron                      ; motor-on status (1 byte, both drives)
        .globl  _acctim                       ; time (200 hz tick) of last access


*------ Imports:
        .globl  flock                         ; floppy/FIFO lock variable
        .globl  _frclock                      ; vbl-frame-counter
        .globl  _nflops                       ; number of floppy drives attached
        .globl  _curflop                      ; currently inserted floppy
        .globl  _critic                       ; critical error handler
        .globl  seekrate                      ; default floppy seek rate
        .globl  _diskmode                     ; disk change mode
        .globl  _hz_200                       ; 200 hz timer ticker


*------ media change modes:
m_changed       equu    2                     ; "CHANGED" media
m_unsure        equ     1                     ; "UNSURE" about media change


*------ Error returns
e_error         equ     -1                    ; general catchall
e_nready        equ     -2                    ; drive-not-ready
e_crc           equ     -4                    ; CRC error
e_seek          equ     -6                    ; seek error
e_rnf           equ     -8                    ; record (sector) not found
e_write         equ     -10                   ; generic write error
e_read          equ     -11                   ; generic read error
e_wp            equ     -13                   ; write on write-protected media
e_badsects      equ     -16                   ; bad sectors on format-track
e_insert        equ     -17                   ; insert_a_disk


*------ Floppy state variables in DSB:
recal           equ     $ff00                 ; recalibrate flag (in dcurtrack)
dcurtrack       equ     0                     ; current track#
dseekrt         equ     dcurtrack+2           ; floppy's seek-rate
dsbsiz          equ     dseekrt+2             ; (size of a DSB)



*--- DMA chip:
diskctl         equ     $ffff8604             ; disk controller data access
fifo            equ     $ffff8606             ; DMA mode control / status
dmahigh         equ     $ffff8609             ; DMA base high
dmamid          equ     $ffff860b             ; DMA base medium
```

```
dmalow          equ     $ffff860d       ; DMA base low

*--- 1770 select values:
cmdreg          equ     $80             ; select command register
trkreg          equ     $82             ; select track register
secreg          equ     $84             ; select sector register
datareg         equ     $86             ; select data register

*--- GI ("psg") sound chip:
giselect        equ     $ffff8800       ; (W) sound chip register select
giread          equ     $ffff8800       ; (R) sound chip read-data
giwrite         equ     $ffff8802       ; (W) sound chip write-data
giporta         equ     $e              ; GI register# for I/O port A

*--- 68901 ("mfp") sticky chip:
mfp     equ     $fffffa00               ; mfp base
gpip    equ     mfp+1                   ; general purpose I/O




*+
*
* SYNOPSIS (synopsisi?):
*
* _flopini(dsb, OL, devno)
* _floprd(dsb, buf, devno, sectno, trackno, sideno, count)
* _flopwr(dsb, buf, devno, sectno, trackno, sideno, count)
* _flopfmt(dsb, buf, devno, spt, trackno, sideno, interlv, magicno, virgin)
* _flopvbl()
* _flopver(dsb, buf, devno, sectno, trackno, sideno, count)
*
* An "EQ" return means success.  Zero is returned in DO.W.
* An "NE" return means failure.  Some negative error number is return in DO.W.
*
* Parameter types (in general):
*       LONG dsb, buf;
*       WORD devno, sectno, trackno, count;
*       WORD spt, interlv, virgin;
*       LONG magicno;
*
*-



*+
* flopini - initialize floppies
* Passed (on the stack):
*       $c(sp) devno
*       $8(sp) ->DSB
*       $4(sp) ->buffer (unused)
*       $0(sp) return address
*
```

```
* Returns:       EQ if initialization succeeded (drive attached).
*                NE if initialization failed (no drive attached).
*-
_flopini:
        lea     dsb0,a1                 ; get ptr to correct DSB
        tst.w   $c(sp)
        beq     fi_1
        lea     dsb1,a1

fi_1:   move.w  seekrate,dseekrt(a1)    ; setup default seek rate
        moveq   #e_error,d0             ; (default error)
        clr.w   dcurtrack(a1)           ; fake clean drive
        bsr     floplock                ; setup parameters
        bsr     select                  ; select drive and side
        move.w  #recal,dcurtrack(a1)    ; default = recal drive (it's dirty)

        bsr     restore                 ; attempt restore
        beq     fi_ok                   ; (quick exit if that won)
        moveq   #10,d7                  ; attempt seek to track 10
        bsr     hseek1                  ; (hard seek to 'd7')
        bne     fi_nok                  ; (failed: drive unusable)
        bsr     restore                 ; attempt restore after seek
fi_ok:  beq     flopok                  ; return OK (on win)
fi_nok: bra     flopfail                ; return failure


*+
* floprd - read sector from floppy
* Passed (on the stack):
*       $14(sp) count
*       $12(sp) sideno
*       $10(sp) trackno
*        $e(sp) sectno
*        $c(sp) devno
*        $8(sp) ->DSB
*        $4(sp) ->buffer
*        $0(sp) return address
*
* Returns:       EQ, the read won (on all sectors),
*                NE, the read failed (on some sector).
*-
_floprd:
        bsr     change                  ; test for disk change
        moveq   #e_read,d0              ; set default error#
        bsr     floplock                ; lock floppies, setup parameters
frd1:   bsr     select                  ; select drive, setup registers
        bsr     go2track                ; seek appropriate track
        bne     frde                    ; retry on seek failure

        move.w  #e_error,curr_err       ; set general error#
        move.w  #$090,(a6)              ; toggle DMA data direction,
        move.w  #$190,(a6)              ;       leave hardware in READ state
        move.w  #$090,(a6)
        move.w  ccount(a5),diskctl      ; set sector count register
        move.w  #$080,(a6)              ; startup 1770 "read sector" command
        move.w  #$90,d7                 ; (read multiple)
        bsr     wdiskctl
        move.l  #timeout,d7             ; set timeout count
        move.l  edma(a5),a2             ; a2 -> target DMA address

*--- Wait for read completion:
frd2:   btst.b  #5,gpip                 ; 1770 done yet?
        beq     frd4                    ; (yes)
        subq.l  #1,d7                   ; decrement timeout counter
        beq     frd3                    ; (punt on timeout)
        move.b  dmahigh,tmpdma+1(a5)    ; get hardware DMA pointer
        move.b  dmamid,tmpdma+2(a5)     ; (most significant bytes FIRST)
        move.b  dmalow,tmpdma+3(a5)
        cmp.l   tmpdma(a5),a2           ; if(tmpdma < edma) continue;
        bgt     frd2

        bsr     reset1770               ; we're done -- interrupt controller
        bra     frd4                    ; see if the read won

*--- timeout: reset the controller and retry:
frd3:   move.w  #e_nready,curr_err(a5)  ; set "timeout" error
        bsr     reset1770               ; (clobber 1770)
        bra     frde                    ; (go retry)

*--- check status after read:
frd4:   move.w  #$090,(a6)              ; examine DMA status register
        move.w  (a6),d0
        btst    #0,d0                   ; bit zero indicates DMA error
        beq     frde                    ; (when its zero -- retry)

        move.w  #$080,(a6)              ; examine 1770 status register
        bsr     rdiskctl
        and.b   #$18,d0                 ; check for RNF, checksum, lost-data
        beq     flopok                  ; return OK if no errors
        bsr     err_bits                ; set error# from 1770 bits
frde:   cmp.w   #midretry,retrycnt(a5)  ; are we on the "middlemost" retry?
        bne     frd5
frde1:  bsr     reseek                  ; yes, home and reseek the head
frd5:   subq.w  #1,retrycnt(a5)         ; drop retry count
        bpl     frd1                    ; (continue if any retries left)
        bra     flopfail                ; fail when we run out of patience


*+
* err_bits - set "curr_err" according to 1770 error status
* Passed:       d0 = 1770 status
*
* Returns:      curr_err, containing current error number
*
* Uses:         d1
*-
err_bits:
        moveq   #e_wp,d1                ; write protect?
        btst    #6,d0
        bne.s   eb1
        moveq   #e_rnf,d1               ; record-not-found?
        btst    #4,d0
        bne.s   eb1

        moveq   #e_crc,d1               ; CRC error?
        btst    #3,d0
        beq     eb1
        move    def_error(a5),d1        ; use default error#
eb1:    move.w  d1,curr_err(a5)         ; set current error number & return
        rts


 ifeq das_boot
*+
* flopwr - write sector to floppy
* Passed (on the stack):
*       $14(sp) count
*       $12(sp) sideno
*       $10(sp) trackno
*        $e(sp) sectno
*        $c(sp) devno
*        $8(sp) ->DSB
*        $4(sp) ->buffer (unused)
*        $0(sp) return address
*
* Returns:      EQ, the write won (on all sectors),
*               NE, the write failed (on some sector).
*-
_flopwr:
        bsr     change                  ; check for disk swap
        moveq   #e_write,d0             ; set default error number
        bsr     floplock                ; lock floppies

*+
* If the boot sector is written to,
* set the media change mode to "unsure".
* (Kludge, kludge, kludge....)
*-
        move.w  csect(a5),d0            ; sector 1
        subq    #1,d0
        or.w    ctrack(a5),d0           ; track 0
        or.w    cside(a5),d0            ; side 0
        bne     fwr1                    ; if not boot sector, then OK
        moveq   #m_changed,d0           ; set media change mode to unsure
        bsr     setdmode                ; (boy, is this /ugly/)

fwr1:   bsr     select                  ; select drive
        bsr     go2track                ; seek
        bne     fwre1                   ; (retry on seek failure)
fwr1a:  move.w  #e_error,curr_err(a5)   ; set general error#
        move.w  #$190,(a6)              ; toggle DMA chip to clear status
        move.w  #$090,(a6)
        move.w  #$190,(a6)              ; leave in WRITE mode
        move.w  #1,d7                   ; load sector-count register
        bsr     wdiskctl
        move.w  #$180,(a6)              ; load "WRITE SECTOR" command
        move.w  #$a0,d7                 ; into 1770 cmdreg
        bsr     wdiskctl
        move.l  #timeout,d7             ; d7 = timeout count

fwr2:   btst.b  #5,gpip                 ; done yet?
        beq     fwr4                    ; (yes, check status)
        subq.l  #1,d7                   ; decrement timeout count
        bne     fwr2                    ; (still tickin')
        bsr     reset1770               ; timed out -- reset 1770
        bra     fwre                    ; and retry

fwr4:   move.w  #$180,(a6)              ; get 1770 status
        bsr     rdiskctl
        bsr     err_bits                ; compute 1770 error bits
        btst    #6,d0                   ; if write protected, don't retry
        bne     flopfail                ; (can't write, so punt)
        and.b   #$5c,d0                 ; check WriteProt+RecNtFnd+CHKSUM+LostD
        bne     fwre                    ; retry on error

        addq.w  #1,csect(a5)            ; bump sector number
        add.l   #$200,cdma(a5)          ; and DMA pointer for next sector
        subq.w  #1,ccount(a5)           ; if(!--count) return OK;
        beq     flopok
        bsr     select1                 ; setup sector#, DMA pointer
        bra     fwr1a                   ; write next (no seek)

fwre:   cmp.w   #midretry,retrycnt(a5)  ; re-seek head in "middle" retry
        bne     fwr5                    ; (not middle retry)
fwre1:  bsr     reseek                  ; home head and seek
fwr5:   subq.w  #1,retrycnt(a5)         ; decrement retry count
        bpl     fwr1                    ; loop if there's still hope
        bra     flopfail                ; otherwise return error status


*+
* _flopfmt - format a track
* Passed (on the stack):
*       $1a(sp) initial sector data
*       $16(sp) magic number
*       $14(sp) interleave
*       $12(sp) side
*       $10(sp) track
*        $e(sp) spt
*        $c(sp) drive
*        $8(sp) pointer to state block
*        $4(sp) dma address
*        $0(sp) [return]
*
* Returns:      EQ: track successfully written.  Zero.W-terminated list of
*               bad sectors left in buffer (they might /all/ be bad.)
*
*               NE: could not write track (write-protected, drive failure,
*               or something catastrophic happened).
*-
_flopfmt:
        cmp.l   #$87654321,$16(sp)      ; check for magic# on stack
        bne     flopfail                ; no magic, so we just saved the world
        bsr     change                  ; check for disk flip
        moveq   #e_error,d0             ; set default error number
        bsr     floplock                ; lock floppies, setup parms
        bsr     select                  ; select drive and side
        move.w  $e(sp),spt(a5)          ; save sectors-per-track
        move.w  $14(sp),interlv(a5)     ; save interleave factor
        move.w  $1a(sp),virgin(a5)      ; save initial sector data

*--- put drive into "changed" mode
        moveq   #m_changed,d0           ; d0 = "CHANGED"
        bsr     setdmode                ; set media change mode

*--- seek to track (hard seek):
        bsr     hseek                   ; hard seek to 'ctrack'
        bne     flopfail                ; (return error on seek failure)
        move.w  ctrack(a5),dcurtrack(a1)        ; record current track#

*--- format track, then verify it:
        move.w  #e_error,curr_err(a5)   ; vanilla error mode
        bsr     fmtrack                 ; format track
        bne     flopfail                ; (return error on seek failure)
        move.w  spt(a5),ccount(a5)      ; set number of sectors to verify
        move.w  #1,csect(a5)            ; starting sector# = 1
        bsr     verify1                 ; verify sectors

*--- if there are any bad sectors, return /that/ error...
        move.l  cdma(a5),a2             ; a2 -> bad sector list
        tst.w   (a2)                    ; any bad sectors?
        beq     flopok                  ; no -- return OK
        move.w  #e_badsects,curr_err(a5) ; set error number
        bra     flopfail                ; return error


*+
* fmtrack - format a track
* Passed:       variables setup by _flopfmt
* Returns:      NE on failure, EQ on success
* Uses:         almost everything
* Called-by:    _flopfmt
*
*-
fmtrack:
        move.w  #e_write,def_error(a5)  ; set default error number
        move.w  #1,d3                   ; start with sector 1, first pass
        move.l  cdma(a5),a2             ; a2 -> prototyping area
        move.w  #60-1,d1                ; 60 x $4e (track leadin)
        move.b  #$4e,d0
        bsr     wmult

*--- address mark
ot3:    move.w  d3,d4                   ; d4 = starting sector (this pass)
ot1:    move.w  #12-1,d1                ; 12 x $00
        clr.b   d0
        bsr     wmult
        move.w  #3-1,d1                 ; 3 x $f5
        move.b  #$f5,d0
        bsr     wmult
        move.b  #$fe,(a2)+              ; $fe -- address mark intro
        move.b  ctrack+1,(a2)+          ; track#
        move.b  cside+1,(a2)+           ; side#
        move.b  d4,(a2)+                ; sector#
        move.b  #$02,(a2)+              ; sector size (512)
        move.b  #$f7,(a2)+              ; write checksum

*--- gap between AM and data:
        move.w  #22-1,d1                ; 22 x $4e
        move.b  #$4e,d0
        bsr     wmult
        move.w  #12-1,d1                ; 12 x $00
        clr.b   d0
        bsr     wmult
        move.w  #3-1,d1                 ; 3 x $f5
        move.b  #$f5,d0
        bsr     wmult

*--- data block:
        move.b  #$fb,(a2)+              ; $fb -- data intro
        move.w  #256-1,d1               ; 256 x virgin.W (initial sector data)
ot2:    move.b  virgin(a5),(a2)+        ; copy high byte
        move.b  virgin+1(a5),(a2)+      ; copy low byte
        dbra    d1,ot2                  ; fill 512 bytes
        move.b  #$f7,(a2)+              ; $f7 -- write checksum
        move.w  #40-1,d1                ; 40 x $4e
        move.b  #$4e,d0
        bsr     wmult

        add.w   interlv(a5),d4          ; bump sector#
        cmp.w   spt(a5),d4              ; if(d4 <= spt) then_continue;
        ble     ot1                     ; proto more sectors this pass
        add.w   #1,d3                   ; bump pass start count
        cmp.w   interlv(a5),d3          ; if(d3 <= interlv) then_continue;
        ble     ot3

*--- end-of-track
        move.w  #1400,d1                ; 1401 x $4e -- end of track trailer
        move.b  #$4e,d0
        bsr     wmult

*--- setup to write the track:
        move.b  cdma+3(a5),dmalow       ; load dma pointer
        move.b  cdma+2(a5),dmamid
        move.b  cdma+1(a5),dmahigh
        move.w  #$190,(a6)              ; toggle R/W flag and
        move.w  #$090,(a6)              ; select sector-count register
        move.w  #$190,(a6)
        move.w  #$1f,d7                 ; (absurd sector count)
        bsr     wdiskctl
        move.w  #$180,(a6)              ; select 1770 cmd register
        move.w  #$f0,d7                 ; write format_track command
        bsr     wdiskctl
        move.l   #timeout,d7                  ; d7 = timeout value

*--- wait for 1770 to complete:
otw1:   btst.b   #5,gpip                      ; is 1770 done?
        beq      otw2                         ; (yes)
        subq.l   #1,d7                        ; if(--d7) continue;
        bne      otw1
        bsr      reset1770                    ; timed out -- reset 1770
oterr:  moveq    #1,d7                        ; return NE (error status)
        rts

*--- see if the write-track won:
otw2:   move.w   #$190,(a6)                   ; check DMA status bit
        move.w   (a6),d0
        btst     #0,d0                        ; if its zero, there was a DMA error
        beq      oterr                        ; (so return NE)
        move.w   #$180,(a6)                   ; get 1770 status
        bsr      rdiskctl
        bsr      err_bits                     ; set 1770 error bits
        and.b    #$44,d0                      ; check for writeProtect & lostData
        rts                                   ; return NE on 1770 error

*------- write 'D1+1' copies of D0.B into A2, A2+1, ...
wmult:  move.b   d0,(a2)+                     ; record byte in proto buffer
        dbra     d1,wmult                     ; (do it again)
        rts



*+
* _flopver - verify sectors on a track
*       $14(sp) count
*       $12(sp) sideno
*       $10(sp) trackno
*        $e(sp) sectno
*        $c(sp) devno
*        $8(sp) ->DSB
*        $4(sp) ->buffer (at least 1K long)
*        $0(sp) return address
*
* Returns:      NULL.W-terminated list of bad sectors in the buffer if D0 == 0,
*               OR some kind of error (D0 < 0).
*
*-
_flopver:
        bsr      change                       ; hack disk change
        moveq    #e_read,d0                   ; set default error#
        bsr      floplock                     ; lock floppies, setup parameters
        bsr      select                       ; select floppy
        bsr      go2track                     ; go to track
        bne      flopfail                     ; (punt if that fails)
        bsr      verify1                      ; verify some sectors
        bra      flopok                       ; return "OK"


*+
* verify1 - verify sectors on a single track
* Passed:           csect = starting sector#
*                   ccount = number of sectors to verify
*                   cdma -> 1K buffer (at least)
*
* Returns:          NULL.W-terminated list of bad sectors (in the buffer)
*                   (buffer+$200..buffer+$3ff used as DMA buffer)
*
* Enviroment:       Head seeked to the correct track;
*                   Drive and side already selected;
*                   Motor should be spinning (go2track and fmttrack do this).
*
* Uses:             Almost everything.
*
* Called-by:        _flopfmt, _flopver
*
*-
verify1:
        move.w  #e_read,def_error(a5)   ; set default error number
        move.l  cdma(a5),a2             ; a2 -> start of bad sector list
        add.l   #$200,cdma(a5)          ; bump buffer up 512 bytes

*--- setup for (next) sector
tvrlp:  move.w  #retries,retrycnt(a5)   ; init sector-retry count
        move.w  #secreg,(a6)            ; load 1770 sector register
        move.w  csect(a5),d7            ; with 'csect'
        bsr     wdiskctl

*--- setup for sector read
tvr1:   move.b  cdma+3(a5),dmalow       ; load dma pointer
        move.b  cdma+2(a5),dmamid
        move.b  cdma+1(a5),dmahigh
        move.w  #$090,(a6)              ; toggle R/W (leave in W state)
        move.w  #$190,(a6)
        move.w  #$090,(a6)
        move.w  #1,d7                   ; set DMA sector count to 1
        bsr     wdiskctl
        move.w  #$080,(a6)              ; load 1770 command register
        move.w  #$80,d7                 ; with ReadSector command
        bsr     wdiskctl
        move.l  #timeout,d7             ; set timeout value

*--- wait for command completion
tvr2:   btst.b  #5,gpip                 ; test for 1770 done
        beq     tvr4                    ; (yes, it completed)
        subq.l  #1,d7                   ; decrement timeout count
        bne     tvr2                    ; (still counting down)
        bsr     reset1770               ; reset controller and return error
        bra     tvre

*--- got "done" interrupt, check DMA status:
tvr4:   move.w  #$090,(a6)              ; read DMA error status
        move.w  (a6),d0
        btst    #0,d0                   ; if DMA_ERROR is zero, then retry
        beq     tvre
```

```
*--- check 1770 completion status (see if it's happy):
        move.w  #$080,(a6)              ; read 1770 status register
        bsr     rdiskctl
        bsr     err_bits                ; set error# from 1770 register
        and.b   #$1c,d0                 ; check for record-not-found, crc-err,
        bne     tvre                    ;       and lost data; return on error

*--- read next sector (or return if done)
tvr6:   addq.w  #1,csect(a5)            ; bump sector count
        subq.w  #1,ccount(a5)           ; while(--count) read_another;
        bne     tvrlp
        sub.l   #$200,cdma(a5)          ; readjust DMA pointer
        clr.w   (a2)                    ; terminate bad sector list
        rts                             ; and return EQ

*--- read failure: retry or record bad sector
tvre:   cmp.w   #midretry,retrycnt(a5)  ; re-seek head?
        bne     tvr5                    ; (no)
        bsr     reseek                  ; yes: back to home and then back
tvr5:   subq.w  #1,retrycnt(a5)         ;       to the current track...
        bpl     tvr1
        move.w  csect(a5),(a2)+         ; record bad sector
        bra.s   tvr6                    ; do next sector
 endc


*+
* _flopvbl - floppy vblank handler
* Deselects floppies after the motor stops.
*-
_flopvbl:
        clr.l   a5                      ; a5 -> zeropage base
        lea     fifo,a6                 ; a6 -> fifo
        st.b    _motoron(a5)            ; assume motor is on
        tst.w   flock(a5)               ; floppies locked?
        bne     fvblr                   ; (yes, so don't touch them)

*------ write-protect monitor:
        move.l  _frclock,d0             ; check a drive every 8 jiffies
        move.b  d0,d1                   ; (save jiffy count)
        and.b   #7,d1                   ; time yet?
        bne     fvbl1                   ; (no)
        move.w  #cmdreg,(a6)            ; select 1770 command/status register

*--- select drive, record it's WP status:
        lsr.b   #3,d0                   ; use bit 4 as drive# to check
        and.w   #1,d0                   ; (keep only bit 0)
        lea     _wpstatus(a5),a0        ; a0 -> write-protect status table
        add.w   d0,a0                   ; a0 -> WP-status table entry

        cmp.w   _nflops,d0              ; if(d0 == _nflops == 1)
        bne     fvbl2                   ;       d0 = 0;
        clr.w   d0
fvbl2:  addq.b  #1,d0                   ; turn into drive-select bits
        lsl.b   #1,d0                   ; (magic shift left)
        eor.b   #7,d0                   ; invert select bits, select side 0
        bsr     setporta                        ; set port A (d2 = old bits)
        move.w  diskctl,d0                      ; get 1770 status
        btst    #6,d0                           ; test Write-Protect status bit
        sne.b   (a0)                            ; set WP status to $00 or $FF.
        move.b  d2,d0                           ; restore old drive-select bits
        bsr     setporta

fvbl1:  move.w  _wpstatus(a5),d0                ; or _wpstatus into _wplatch
        or.w    d0,_wplatch(a5)                 ; (catch any WP transitions)


*------ floppy deselect test:
        tst.w   deselflg(a5)                    ; floppies already deselected?
        bne     fvblr1                          ; (yes, so don't do it again)

        bsr     rdiskctl                        ; read 1770 status register
        btst    #7,d0                           ; is the motor still on?
        bne     fvblr                           ; (yes, so don't deselect)
        move.b  #7,d0                           ; deselect both drives
        bsr     setporta                        ; (set bits 0..3 in portA of PSG)
        move.w  #1,deselflg(a5)                 ; indicate floppies deselected
fvblr1: clr.w   _motoron(a5)                    ; indicate motor is OFF
fvblr:  rts                                     ; back to vbl



*+
* floplock - lock floppies and setup floppy parameters
*
* Passed (on the stack):
*       $18(sp) - count.W (sector count)
*       $16(sp) - side.W (side#)
*       $14(sp) - track.W (track#)
*       $12(sp) - sect.W (sector#)
*       $10(sp) - dev.W (device#)
*        $c(sp) - obsolete.L
*         8(sp) - dma.L (dma pointer)
*         4(sp) - ret1.L (caller's return address)
*         0(sp) - ret.L (floplock's return address)
*
* Passed:       D0.W = default error number
*-
floplock:
        movem.l d3-d7/a3-a6,regsave             ; save C registers

        clr.l   a5                              ; a5 -> zeropage base
        lea     fifo,a6                         ; a6 -> fifo
        st      _motoron                        ; kludge motor state = ON
        move.w  d0,def_error(a5)                ; set default error number
        move.w  d0,curr_err(a5)                 ; set current error number
        move.w  #1,flock(a5)                    ; tell vbl not to touch floppies
        move.l  8(sp),cdma(a5)                  ; cdma -> /even/ DMA address
        move.w  $10(sp),cdev(a5)                ; save device# (0..1)
        move.w  $12(sp),csect(a5)               ; save sector# (1..9, usually)
        move.w  $14(sp),ctrack(a5)              ; save track# (0..39..79   )
        move.w  $16(sp),cside(a5)               ; save side# (0..1)
        move.w  $18(sp),ccount(a5)      ; save sector count (1..spt)
        move.w  #retries,retrycnt(a5)   ; setup retry count

*--- pick a DSB:
        lea     dsb0(a5),a1
        tst.w   cdev(a5)
        beq     flock2
        lea     dsb1(a5),a1

*--- compute ending DMA address from count parameter:
flock2: moveq   #0,d7
        move.w  ccount(a5),d7           ; edma = cdma + (ccount * 512)
        lsl.w   #8,d7
        lsl.w   #1,d7
        move.l  cdma(a5),a0
        add.l   d7,a0
        move.l  a0,edma(a5)

*--- recalibrate drive (if it needs it)
        tst.w   dcurtrack(a1)           ; if (curtrack < 0) recalibrate()
        bpl     flockr

        bsr     select                  ; select drive & side
        clr.w   dcurtrack(a1)           ; we're optimistic -- assume winnage
        bsr     restore                 ; attempt restore
        beq     flockr                  ; (it won)
        moveq   #10,d7                  ; attempt seek to track 10
        bsr     hseek1
        bne     flock1                  ; (failed)
        bsr     restore                 ; attempt restore again
        beq     flockr                  ; (it won)
flock1  move.w  #recal,dcurtrack(a1)    ; complete failure (what can we do?)
flockr: rts


*+
* flopfail - unlock floppies and return error.
*
*-
flopfail:
        moveq   #m_unsure,d0            ; disk change mode = UNSURE
        bsr     setdmode                ; set media change mode
        move.w  curr_err(a5),d0         ; get current error number
        ext.l   d0                      ; extend to long
        bra.s   unlok1                  ; clobber floppy lock & return

*+
* flopok - unlock floppies and return success status:
*
*-
flopok: clr.l   d0                      ; return 0 (success)
unlok1: move.l  d0,-(sp)                ; (save return value)
        move.w  #datareg,(a6)           ; force WP to real-time mode
        move.w  dcurtrack(a1),d7        ; dest-track = current track
        bsr     wdiskctl
        move.w  #$10,d6                 ; cmd = seek w/o verify
        bsr     flopcmds                ; do it

        move.w  cdev,d0                 ; set last-access time for 'cdev'
        lsl.w   #2,d0
        lea     _acctim,a0
        move.l  _hz_200(a5),(a0,d0.w)
        cmp.w   #1,_nflops              ; if (nflops == 1) set other time, too
        bne     unlok2
        move.l  _hz_200(a5),4(a0)       ; set last-accessed time for floppy 1

unlok2: move.l  (sp)+,d0                ; restore return value
        movem.l regsave,d3-d7/a3-a6     ; restore C registers
        clr.w   flock                   ; unlock floppies
        rts



*+
* hseek  - seek to 'ctrack' without verify
* hseek1 - seek to 'd7' without verify
* hseek2 - seek to 'd7' without verify, keep current error number
*
* Returns:      NE on seek failure ("cannot happen"?)
*               EQ if seek wins
*
* Uses:         d7, d6, ...
* Jumps-to:     flopcmds
* Called-by:    _flopfmt, _flopini
*
*-
hseek:  move.w  ctrack,d7               ; dest track = 'ctrack'
hseek1: move.w  #e_seek,curr_err        ; possible error = "seek error"
hseek2: move.w  #datareg,(a6)           ; write destination track# to data reg
        bsr     wdiskctl
        move.w  #$10,d6                 ; execute "seek" command
        bra     flopcmds                ; (without verify...)


*+
* reseek - home head, then reseek track
* Returns:      EQ/NE on success/failure
* Falls-into:   go2track
*
*-
reseek:
        move.w  #e_seek,curr_err        ; set "seek error"
        bsr     restore                 ; restore head
        bne     go2trr                  ; (punt if home fails)

        clr.w   dcurtrack(a1)           ; current track = 0
        move.w  #trkreg,(a6)            ; set "current track" reg on 1770
        clr.w   d7
        bsr     wdiskctl

        move.w  #datareg,(a6)           ; seek out to track five
        move.w  #5,d7
        bsr     wdiskctl                ; dest track = 5
        move.w  #$10,d6
        bsr     flopcmds                ; seek
        bne     go2trr                  ; return error on seek failure
        move.w  #5,dcurtrack(a1)        ; set current track#

*+
* go2track - seek proper track
* Passed:       Current floppy parameters (ctrack, et al.)
* Returns:      EQ/NE on success/failure
* Calls:        flopcmds
*-
go2track:
        move.w  #e_seek,curr_err        ; set "seek error"
        move.w  #datareg,(a6)           ; set destination track# in
        move.w  ctrack(a5),d7           ;  1770's data register
        bsr     wdiskctl                ; (write track#)
        moveq   #$14,d6                 ; execute 1770 "seek_with_verify"
        bsr     flopcmds                ; (include seek-rate bits)
        bne     go2trr                  ; return error on seek failure
        move.w  ctrack(a5),dcurtrack(a1) ; update current track number
        and.b   #$18,d7                 ; check for RNF, CRC_error, lost_data
go2trr: rts                             ; return EQ/NE on succes/failure



*+
* restore - home head
* Passed:       nothing
* Returns:      EQ/NE on success/failure
* Falls-into:   flopcmds
*-
restore:
        clr.w   d6                      ; $00 = 1770 "restore" command
        bsr     flopcmds                ; do restore
        bne     res_r                   ; punt on timeout
        btst    #2,d7                   ; test TRK00 bit
        eor     #$04,ccr                ; flip Z bit (return NE if bit is zero)
        bne     res_r                   ; punt if didn't win
        clr.w   dcurtrack(a1)           ; set current track#
res_r:  rts



*+
* flopcmds - floppy command (or-in seek speed bits from database)
* Passed:       d6.w = 1770 command
* Sets-up:      seek bits (bits 0 and 1) in d6.w
* Falls-into:   flopcmd
* Returns:      EQ/NE on success/failure
*-
flopcmds:
        move.w  dseekrt(a1),d0          ; get floppy's seek rate bits
        and.b   #3,d0                   ; OR into command
        or.b    d0,d6
```

```
*+
* flopcmd - execute 1770 command (with timeout)
* Passed:        d6.w = 1770 command
*
* Returns:       EQ/NE on success/failure
*                d7 = 1770 status bits
*
*-
flopcmd:
        move.l  #timeout,d7             ; setup timeout count (assume short)
        move.w  #cmdreg,(a6)            ; select 1770 command register
        bsr     rdiskctl                ; read it to clobber READY status
        btst    #7,d0                   ; is motor on?
        bne     flopcm                  ; (yes, keep short timeout)    .
        move.l  #ltimeout,d7            ; extra timeout for motor startup
flopcm: bsr     wdiskct6                ; write command (in d6)

flopcl: subq.l  #1,d7                   ; timeout?
        beq     flopcto                 ; (yes, reset and return failure)
        btst.b  #5,gpip                 ; 1770 completion?
        bne     flopcl                  ; (not yet, so wait some more)
        bsr     rdiskct7                ; return EQ + 1770 status in d7
        clr.w   d6
        rts
flopcto:
        bsr     reset1770               ; bash controller
        moveq   #1,d6                   ; and return NE
        rts


*+
* reset1770 - reset disk controller after a catastrophe
* Passed:        nothing
* Returns:       nothing
* Uses:          d7
*-
reset1770:
        move.w  #cmdreg,(a6)            ; execute 1770 "reset" command
        move.w  #$d0,d7
        bsr     wdiskctl
        move.w  #15,d7                  ; wait for 1770 to stop convulsing
r1770:  dbra    d7,r1770                ; (short delay loop)
        bsr     rdiskct7                ; return 1770 status in d7
        rts


*+
* select - setup drive select, 1770 and DMA registers
* Passed:        cside, cdev
* Returns:       appropriate drive and side selected
*-
select:
        clr.w   deselflg(a5)            ; floppies NOT deselected
        move.w  cdev(a5),d0             ; get device number
        addq.b  #1,d0                   ; add and shift to get select bits
        lsl.b   #1,d0                   ; into bits 1 and 2
        or.w    cside(a5),d0            ; or-in side number (bit 0)
        eor.b   #7,d0                   ; negate bits for funky hardware select
        and.b   #7,d0                   ; strip anything else out there
        bsr     setporta                ; do drive select

        move.w  #trkreg,(a6)            ; setup 1770 track register
        move.w  dcurtrack(a1),d7        ;       from current track number
        bsr     wdiskctl
        clr.b   tmpdma(a5)              ; zero bits 24..32 of target DMA addr

*--- alternate entry point: setup R/W parameters on 1770
select1:
        move.w  #secreg,(a6)            ; setup requested sector_number from
        move.w  csect(a5),d7            ;       caller's parameters
        bsr     wdiskctl
        move.b  cdma+3(a5),dmalow       ; setup DMA chip's DMA pointer
        move.b  cdma+2(a5),dmamid
        move.b  cdma+1(a5),dmahigh
        rts


*+
* setporta - set floppy select bits in PORT A on the sound chip
* Passed:       d0.b (low three bits)
* Returns:      d1 = value written to port A
*               d2 = old value read from port A
* Uses:         d1
*-
setporta:
        move    sr,-(sp)                ; save our IPL
        or      #$0700,sr               ; start critical section
        move.b  #giporta,giselect       ; select port on GI chip
        move.b  giread,d1               ; get current bits
        move.b  d1,d2                   ; save old bits for caller
        and.b   #$ff-7,d1               ; strip low three bits there
        or.b    d0,d1                   ; or-in our new bits
        move.b  d1,giwrite              ; and write 'em back out there
        move    (sp)+,sr                ; restore IPL to terminate CS, return
        rts


*+
* Primitives to read/write 1770 controller chip (DISKCTL register).
*
* The 1770 can't keep up with full-tilt CPU accesses, so
* we have to surround reads and writes with delay loops.
* This is not really as slow as it sounds.
*
*-
wdiskct6:                               * write d6 to diskctl
        bsr     rwdelay                 ;       delay
        move.w  d6,diskctl              ;       write it
        bra     rwdelay                 ;       delay and return

wdiskctl:                               * write d7 to diskctl
        bsr     rwdelay                 ;       delay
        move.w  d7,diskctl              ;       write it
        bra     rwdelay                 ;       delay and return

rdiskct7:                               * read diskctl into d7
        bsr     rwdelay                 ;       delay
        move.w  diskctl,d7              ;       read it
        bra     rwdelay                 ;       delay and return

rdiskctl:                               * read diskctl into d0
        bsr     rwdelay                 ;       delay
        move.w  diskctl,d0              ;       read it
rwdelay:
        move    sr,-(sp)                ; save flags
        move.w  d7,-(sp)                ; save counter register
        move.w  #$20,d7                 ; 0x20 seems about right...
rwdly1: dbra    d7,rwdly1               ; busy-loop: give 1770 time to settle
        move.w  (sp)+,d7                ; restore register, flags, and return
        move    (sp)+,sr
        rts


*+
* change - check to see if the "right" floppy has been inserted
* On the stack:
*       $10(sp) - dev.W (device#)
*        $c(sp) - dsb.L (pointer to Device State Block)
*         8(sp) - dma.L (dma pointer)
*         4(sp) - ret1.L (caller's return address)
*         0(sp) - ret.L (change's return address)
*
* Returns:      both media "might have changed" condition
*
* Uses:         C registers
*
*-
change:
        cmp.w   #1,_nflops              ; if there are zero or two floppies
        bne     ch_r                    ;       then do nothing (return OK)
        move.w  $10(sp),d0              ; if cdev == _curflop
        cmp.w   _curflop,d0             ; (...current disk == current drive?)
        beq     ch_ok1                  ; then return OK (but use drive #0)

*--- ask the user to stick in the other floppy (via critical error handler)
        move.w  d0,-(sp)                ; push disk# we want inserted
        move.w  #e_insert,-(sp)         ; push "INSERT_A_DISK" error number
        bsr     _critic                 ; use critical error handler and
        add.w   #4,sp                   ;       hope somebody handles it
        move.w  #$ffff,_wplatch         ; set "might have changed" on both drvs
        move.w  $10(sp),_curflop        ; set current disk#
ch_ok1: clr.w   $10(sp)                 ; use drive 0
ch_r:   rts


*+
* setdmode - set drive-change mode
* Passed:       d0.b = mode to put current drive in (0, 1, 2)
* Uses:         a0
*
*-
setdmode:
        lea     _diskmode,a0            ; a0 -> disk mode table
        move.b  d0,-(sp)                ; (save mode)
        move.w  cdev(a5),d0             ; d0.w = drive# (index into table)
        move.b  (sp)+,(a0,d0.w)         ; set drive's mode
        rts



__dskf: dc.b %10101110
        dc.b %11010110
        dc.b %10001100
        dc.b %00010111
        dc.b %11111011
        dc.b %10000000
        dc.b %01101010
        dc.b %00101011
        dc.b %10100110
        even


*-------------- Floppy RAM usage:
        bss
retrycnt:       ds.w    1               ; retry counter         (used)
_wpstatus:      ds.b    2               ; WP status (2 drives)  status
_wplatch:       ds.b    2               ; WP latch (2 drives)   status
_acctim:        ds.l    2               ; last access counter
_motoron:       ds.w    1               ; motor-on-P (both drives)  status
deselflg:       ds.w    1               ; deselect flag         state

cdev:           ds.w    1               ; device #              parm
ctrack:         ds.w    1               ; track number          parm
csect:          ds.w    1               ; sector number         parm
cside:          ds.w    1               ; side number           parm
ccount:         ds.w    1               ; sector count          parm
cdma:           ds.l    1               ; DMA address           parm
edma:           ds.l    1               ; ending DMA address    computed

spt:            ds.w    1               ; #sectors_per_track    flopfmt parm
interlv:        ds.w    1               ; interleave factor     flopfmt parm
virgin:         ds.w    1               ; fill data for sectors flopfmt parm

tmpdma:         ds.l    1               ; temp for hardware DMA image
def_error:      ds.w    1               ; default error number
curr_err:       ds.w    1               ; current error number

regsave:        ds.l    9               ; save area for C registers
dsb0:           ds.b    dsbsiz          ; floppy 0's DSB
dsb1:           ds.b    dsbsiz          ; floppy 1's DSB
```

```
************************************************************************
*                                                                      *
*               ST SERIES BIOS SOURCE REV.  A                          *
*               THIS PORTION BY D.  GETREU                             *
*                                                                      *
*                                                                      *
*               copyright 1984,1985 atari corporation                  *
*               all rights reserved                                    *
*                                                                      *
*                                                                      *
************************************************************************
*+
* rbios.s - character I/O routines
*
* Oct-Feb 84/85 dbg     Backed it up
* 13-Mar-1985 lmd       Ripped out 'conout' (now in escape.s)
* may 7,1985 dbg        conditional assembly added for country of origin
*                       (USA, UK/ITALY, GERMANY, FRANCE)
*
*-

*+ (lmd)
* Imports:
*
*-
        .globl  _timr_ms                ;timer C calibration
        .globl  etv_timer               ;system timer handoff vector
        .globl  _hz_200                 ;timer c raw tick
        .globl  conterm                 ;console configuration byte
        .globl  _dumpflg                ;flag to signal a screen dump(alt-HELP)


*+ (dbg)
* Exports:
*
*-
        .globl  kbshift
        .globl  pconfig


USA     equ     0
UK      equ     1
GERMANY equ     2
FRANCE  equ     3

COUNTRY equ     USA     ;set country of origin to USA
*COUNTRY        equ     UK      ;set country of origin to UK
*COUNTRY        equ     GERMANY ;set country of origin to GERMANY
*COUNTRY        equ     FRANCE  ;set country of origin to FRANCE


************************************************************************
*                                                                      *
*               general equates for the rbp system rom                 *
*                                                                      *
************************************************************************
```

```
************************************************************************
*                                                                      *
*                       acia register commands                         *
*                                                                      *
************************************************************************

rsetacia        equ     %00000011       ;reset acia
div64           equ     %00000010       ;set to clock line to /64
div16           equ     %00000001       ;set to clock line to /16

* note the keyboard and midi units expect 8 bits/1 stop bit/no parity!!

protocol        equ     %00010100       ;set to 8 bit/1 stop/no parity

* note the keyboard and midi units may allow for transmitting interrupts
*       therefore we define all possible states here.  we will
*       assume that it is init'ed as bar/rts=low,disabled.

rtsld           equ     %00000000       ;rts=low, interrupt disabled
rtsle           equ     %00100000       ;rts=low, interrupt enabled
rtshd           equ     %01000000       ;rts=high, interrupt disabled
rtsbrk          equ     %01100000       ;rts=low, interrupt disabled, break

* note the keyboard and midi units may be allowed to
*       send interrupts to the host

intron          equ     %10000000       ;interrupts enabled
introff         equ     %00000000       ;interrupts disabled


************************************************************************
*               acia status definitions
************************************************************************

rdrf            equ     %00000001
tdre            equ     %00000010
dcd             equ     %00000100
cts             equ     %00001000
fe              equ     %00010000
ovrn            equ     %00100000
pe              equ     %01000000
irq             equ     %10000000

*       control register "or" mask settings

c19200  equ     1
c9600   equ     1
c4800   equ     1
c3600   equ     1
c2400   equ     1
c2000   equ     1
c1800   equ     1
c1200   equ     1
c600    equ     1
c300    equ     1
```

```
c200    equ     1
c150    equ     1
c134    equ     1
c110    equ     1
c75     equ     2
c50     equ     2

*       timer data register settings

d19200  equ     1
d9600   equ     2
d4800   equ     4
d3600   equ     5               ;3840 baud -- % error of 6.66
d2400   equ     8
d2000   equ     10              ;1920 baud -- % error of 4.00
d1800   equ     11              ;1745 baud -- % error of 2.50
d1200   equ     16
d600    equ     32
d300    equ     64
d200    equ     96
d150    equ     128
d134    equ     143             ;134.26 baud -- % error of 0.19
d110    equ     175             ;109.71 baud -- % error of 0.26
d75     equ     64
d50     equ     96

*************************************************************************
*                                                                       *
*       g.i. sound chip ay-3-8910 definitions and init code             *
*                                                                       *
*************************************************************************

gibase  equ     $ffff8800

*       gi chip register offsets

giselect        equ     gibase+0        ;write data register    word
rddata          equ     gibase+0        ;byte of register       word
wrdata          equ     gibase+2        ;byte of register       word

*       gi register select offset numbers

toneaf  equ     0
toneac  equ     1
tonebf  equ     2
tonebc  equ     3
tonecf  equ     4
tonecc  equ     5
noise   equ     6
mixer   equ     7
aamplt  equ     8
bamplt  equ     9
camplt  equ     10
fienvlp equ     11
crenvlp equ     12
shenvlp equ     13
```

```
porta   equ     14
*
*       port a  - outputs all!
*
*       d0 - side select
*       d1 - drive select 0
*       d2 - drive select 1
*       d3 - rts for rs-232
*       d4 - dtr for rs-232
*       d5 - centronics strobe
*       d6 - general purpose output
*       d7 - unassigned output
*
portb   equ     15      ;parallel i/o port


************************************************************************
*                                                                      *
*       68901 multifunction peripheral chip equates                    *
*       (interrupt controller,timers,serial i/o)                       *
*                                                                      *
************************************************************************

*       register and base addresses

mfp     equ     $fffffa01                      ;base address, +1 offset !!!!!!!!

*       system interrupt register offsets

gpip    equ     0                              ;general purpose i/o
aer     equ     2                              ;active edge register
ddr     equ     4                              ;data direction register
iera    equ     6                              ;interrupt enable register a
ierb    equ     8                              ;interrupt enable register b
ipra    equ     10                             ;interrupt pending register a
iprb    equ     12                             ;interrupt pending register b
isra    equ     14                             ;interrupt in-service register a
isrb    equ     16                             ;interrupt in-service register b
imra    equ     18                             ;interrupt mask register a
imrb    equ     20                             ;interrupt mask register b
vr      equ     22                             ;vector register

*       system timer registers offsets

tacr    equ     24                             ;timer a control register
tbcr    equ     26                             ;timer b control register
tcdcr   equ     28                             ;timer c and d control register
tadr    equ     30                             ;timer a data register
tbdr    equ     32                             ;timer b data register
tcdr    equ     34                             ;timer c data register
tddr    equ     36                             ;timer d data register

*       rs232/rs422/async/sync serial i/o registers offsets

scr     equ     38                             ;sync character register
```

```
ucr     equ     40                      ;usart control register
rsr     equ     42                      ;receiver status register
tsr     equ     44                      ;transmitter status register
udr     equ     46                      ;usart data register


*       non-memory oriented equates for the rs232 port and timers

ctrls   equ     $13                     ;control s
ctrlq   equ     $11                     ;control q
xoff    equ     $13
xon     equ     $11
xonoff  equ     1                       ;used to indicate xon/xoff protocol

*       timer relative locations

atimer  equ     0
btimer  equ     1
ctimer  equ     2
dtimer  equ     3


*************************************************************************
*                                                                       *
*       last modified   9/17/84                                         *
*       created 9/04/84                                                 *
*       by      david b. getreu                                         *
*                                                                       *
*       the following is the acia definitions for the keyboard         *
*       and midi interfacing.  the baud rate for the keyboard acia is   *
*       an amazing 7812.5, a new exciting industrial standard.          *
*       anyways, the appropriate chip setting for this acia is /64,     *
*       while that of the midi interface is /16.  it's baud rate is an  *
*       amazing 31250, another new exciting industrial standard.  the   *
*       500 khz signal to the acia comes off of the glue chip to both   *
*       the keyboard and midi acia tx/rx clocks.                        *
*                                                                       *
*                                                                       *
*************************************************************************

keyboard        equ     $fffffc00       ;keyboard acia address base
midi            equ     $fffffc04       ;midi acia address base

*       register offsets for acias'

comstat equ     0               ;command/status registers
iodata  equ     2               ;keyboard data register


*************************************************************************
*                       ascii character definitions                     *
*************************************************************************

nul     equ     $00
soh     equ     $01
stx     equ     $02
```

```
etx     equ     $03
eot     equ     $04
enq     equ     $05
ack     equ     $06
bel     equ     $07
bs      equ     $08
ht      equ     $09
lf      equ     $0a
vt      equ     $0b
ff      equ     $0c
cr      equ     $0d
so      equ     $0e
si      equ     $0f
dle     equ     $10
dc1     equ     $11
dc2     equ     $12
dc3     equ     $13
dc4     equ     $14
nak     equ     $15
syn     equ     $16
etb     equ     $17
can     equ     $18
em      equ     $19
eof     equ     $1a     ;really 'sub' in ANSI ascii
esc     equ     $1b
fs      equ     $1c
gs      equ     $1d
rs      equ     $1e
us      equ     $1f
spc     equ     $20
del     equ     $7f

************************************************************************
*       exception vector assignment table equates and functions        *
************************************************************************

evsetsp equ     $00     ;power-on reset supervisor stack pointer
evsetpc equ     $04     ;power-on reset initial program counter
buserr  equ     $08     ;bus error
adrerr  equ     $0C     ;address error
illins  equ     $10     ;illegal instruction
zerodiv equ     $14     ;zero divide
chkinst equ     $18     ;chk instruction
trapvf  equ     $1C     ;trap on overflow
privldg equ     $20     ;priviledged instruction
trace   equ     $24     ;trace mode
lin1010 equ     $28     ;line 1010 emulator
lin1111 equ     $2C     ;line 1111 emulator
uninit  equ     $3C     ;uninitialized interrupt vector
spurint equ     $60     ;spurious interrupt
hblank  equ     $68     ;horizontal blank interrupt
vblank  equ     $70     ;vertical blank interrupt
trap0   equ     $80     ;trap instruction 0
trap1   equ     $84     ;trap instruction 1
trap2   equ     $88     ;trap instruction 2
trap3   equ     $8C     ;trap instruction 3
```

```
trap4   equ     $90     ;trap instruction 4
trap5   equ     $94     ;trap instruction 5
trap6   equ     $98     ;trap instruction 7
trap7   equ     $9C     ;trap instruction 7
trap8   equ     $A0     ;trap instruction 8
trap9   equ     $A4     ;trap instruction 9
trap10  equ     $A8     ;trap instruction 10
trap11  equ     $AC     ;trap instruction 11
trap12  equ     $B0     ;trap instruction 12
trap13  equ     $B4     ;trap instruction 13
trap14  equ     $B8     ;trap instruction 14
trap15  equ     $BC     ;trap instruction 15


*************************************************************************
*       interrupt priority table                                        *
*************************************************************************
*                                                                       *
*       priority        vector          description                     *
*       --------        -------         -----------                     *
*       0  low          00_0100 *       centronics busy         i0      *
*       1               00_0104         data carrier detect     i1      *
*       2               00_0108 *       clear-to-send           i2      *
*       3               00_010c         gpu blt done            i3      *
*       4               00_0110         baud rate generator     (d)     *
*       5               00_0114 *       system timer            (c)     *
*       6               00_0118 *       midi/keyboard acia      i4      *
*       7               00_011c         disk dma                i5      *
*       8               00_0120         horizontal blank counter (b)    *
*       9               00_0124 *       tx error                        *
*       10              00_0128 *       tx buffer empty                 *
*       11              00_012c *       receive error                   *
*       12              00_0130 *       receive buffer full             *
*       13              00_0134         user/application timer  (a)     *
*       14              00_0138         ringer indicator        i6      *
*       15 high         00_013c         monochrome detect       i7      *
*************************************************************************

prtint  equ     $100    ;centronics busy                        (i0)
dcd232  equ     $104    ;dcd rs-232 interrupt vector            (i1)
cts232  equ     $108    ;cts rs-232 interrupt vector            (i2)
bltdon  equ     $10C    ;graphics blt done interrupt            (i3)
baudrg  equ     $110    ;baud rate generator interrupt          timer d
unused  equ     $114    ;system clock interrupt                 timer c
midkey  equ     $118    ;midi/keyboard interrupt                (i4)
dskdma  equ     $11C    ;disk dma interrupt                     (i5)
hblnkc  equ     $120    ;horizontal blank counter               timer b
txderr  equ     $124    ;transmitter error interrupt
txbufe  equ     $128    ;transmitter buffer empty interrupt
rxderr  equ     $12C    ;receiver error interrupt
rxbufe  equ     $130    ;receiver buffer full interrupt
sysclk  equ     $134    ;free...free...free...                  timer a
rng232  equ     $138    ;ring indicator rs-232                  (i6)
monitr  equ     $13C    ;monochrome monitor detect              (i7)
```

```
**********************************************************************
*               operating system memory space                        *
**********************************************************************
*
*       rs-232/midi/keyboard rffset equates for their i/o buffer records
*

ibufptr         equ     0       ;input buffer location pointer
ibufsiz         equ     4       ;maximum size of this buffer
ibufhead        equ     6       ;relative pointer to next byte to be taken from
*                               ;this buffer
ibuftail        equ     8       ;relative pointer to next location available to
*                               ;insert a new byte
ibuflow         equ     10      ;amount of space in buffer before an "xon" may
*                               ;be sent to restore normal use of buffer.
ibufhigh        equ     12      ;amount of space used in buffer that trigger's
*                               ;the sending of a "xoff" signal to the host
obufptr         equ     14      ;buffer location pointer
obufsiz         equ     18      ;maximum size of this buffer
obufhead        equ     20      ;relative pointer to next byte to be taken from
*                               ;this buffer
obuftail        equ     22      ;relative pointer to next location available to
*                               ;insert a new byte
obuflow         equ     24      ;amount of space in buffer before an "xon" may
*                               ;be sent to restore normal use of buffer.
obufhigh        equ     26      ;amount of space used in buffer that trigger's
*                               ;the sending of a "xoff" signal to the host
status          equ     28      ;copy of midi acia status
rsrbyte         equ     28      ;copy of rs-232 receiver status byte
tsrbyte         equ     29      ;copy of rs-232 transmitter status byte
rxoff           equ     30      ;rs-232 receiver xoff flag
txoff           equ     31      ;rs-232 transmitter xoff flag
rsmode          equ     32      ;rs-232 control mode

        .bss


rinsize         equ     $100            ;these are size equates, not location
routsize        equ     $100            ;these are size equates, not location

ribuffer        ds.b    rinsize         ;rs-232 input buffer
robuffer        ds.b    routsize        ;rs-232 output buffer

kinsize         equ     $80

kibuffer        ds.b    kinsize         ;keyboard input buffer

minsize         equ     $80

mibuffer        ds.b    minsize         ;midi input buffer

*
*       mfp rs232 port routines variable space
*

ribufptr        ds.l    1
```

```
ribufsiz        ds.w    1
ribufhead       ds.w    1
ribuftail       ds.w    1
ribuflow        ds.w    1
ribufhigh       ds.w    1
robufptr        ds.l    1
robufsiz        ds.w    1
robufhead       ds.w    1
robuftail       ds.w    1
robuflow        ds.w    1
robufhigh       ds.w    1
rrsrbyte        ds.b    1
rtsrbyte        ds.b    1
rrxoff          ds.b    1
rtxoff          ds.b    1
rrsmode         ds.b    2
rbufrec         equ     ribufptr


*
*       keyboard rs232 port routines variable space
*

kibufptr        ds.l    1
kibufsiz        ds.w    1
kibufhead       ds.w    1
kibuftail       ds.w    1
kibuflow        ds.w    1
kibufhigh       ds.w    1
kbufrec         equ     kibufptr


*
*       midi rs232 port routines variable space
*

mibufptr        ds.l    1
mibufsiz        ds.w    1
mibufhead       ds.w    1
mibuftail       ds.w    1
mibuflow        ds.w    1
mibufhigh       ds.w    1
mbufrec         equ     mibufptr


*       Acia error handler vectors -- init'ed to point to 'rte' unless
*       changed subsequent to boot-up

midivec         ds.l    1       ;midi interrupt handler vector
vkbderr         ds.l    1       ;keyboard error handler address
vmiderr         ds.l    1       ;midi error handler address
statintvec      ds.l    1       ;general ikbd status record interrupt vector
msintvec        ds.l    1       ;mouse interrupt vector
clkintvec       ds.l    1       ;ikbd real-time clock interrupt vector
joyintvec       ds.l    1       ;general joystick interrupt vector
```

```
*
*       real-time clock command equates
*
settod  equ     $1b
gettod  equ     $1c

*
*       kstate (ikbd's general state variable) values
*

normal  equ     0
statks  equ     1
amouse  equ     2
rmouse  equ     3
clock   equ     4
joyall  equ     5
joy0    equ     6
joy1    equ     7

*
*       array lengths for ikbd subsystem records
*

statdex equ     7
amdex   equ     5
rmdex   equ     3
clkdex  equ     6
joyadex equ     2
joydex  equ     1

kstate          ds.b    1       ;present state of ikbd reception routine
kindex          ds.b    1       ;index used to count down bytes left to
*                               ;receive for current state's record
statrec         ds.b    statdex
amrec           ds.b    amdex
mousebuf        ds.b    rmdex
clkrec          ds.b    clkdex
joyrec          ds.b    joyadex

datetime        ds.l    1       ;jdos variable
newtime         ds.l    1       ;jdos variable
oclkrec         ds.b    clkdex  ;used to assemble and send a new t.o.d. record
*                               ;to the ikbd

on      equ     1
off     equ     0

kmbuf           ds.b    3       ;key-emulating mouse buffer

* bit assignments in kbshift

KBRSH   EQU     0               * right shift
KBLSH   EQU     1               * left shift
KBCTL   EQU     2               * control key
KBALT   EQU     3               * alternate key
KBCL    EQU     4               * caps lock
```

```
KBMRB   EQU     5               * right mouse button (clr/home)
KBMLB   EQU     6               * left mouse button (insert)

kbshift         ds.b    1

initsize        equ     kbshift-kstate-1        ;area to be inited to zero!

skeytran        ds.l    1       ;contains address for unshifted key translation
skeyshif        ds.l    1       ;contains address for shifted key translation
skeycl          ds.l    1       ;contains address for caps-lock key translation

*       mouse init transfer string buffer

transbuf        ds.b    17      ;temporary string buffer for mouse init's

*       keyrepeat variables

timerate        equ     200     ;timer c rate in Hz.

keyrep          ds.b    1
kdelay1         ds.b    1       ;must start on word boundary
kdelay2         ds.b    1
cdelay1         ds.b    1       ;must start on word boundary
cdelay2         ds.b    1
tdelay1         equ     15      ;delay before key repeat engages
tdelay2         equ     2       ;delay before key repeats after
*                               ;key repeat is activated

*       parallel timeout counter

prt_to          ds.l    1

tc_rot          ds.w    1       ;divisor byte for timer c interrupt

*
*       Dave Staugas' Sound Driver variables
*
cursnd          ds.l    1
timer           ds.b    1
auxd            ds.b    1

*
*       printer configuration word
*
*       bits 6-15 not defined
*
*       bit 5   -       printer uses (_FORMFEED/SINGLE SHEET)
*       bit 4   -       port to send output to (_ATARI/EPSON)
*       bit 3   -       style of output (_DRAFT/FINAL)
*       bit 2   -       type of printer (_DOT MATRIX/DAISY WHEEL)
*       bit 1   -       type of ink (_MONOCHROME/COLOR)
*       bit 0   -       manufacturer (_ATARI/EPSON COMPATIBLE)
*
*       note all underscored settings are the default and are represented
*       by their corresponding bit set to "0"
```

```
pconfig         ds.w    1

*       console and terminal enable flags
*       bit 0 - keyclick enabled
*       bit 1 - repeat key function enabled
*       bit 2 - keyboard "^g" bell feature enabled

*conterm                        ds.b    1       ;now in landon's equates

newtod  ds.b    1       ;handshaking flag for get time of day function


        page
        even
        text

***********************************************************************
*                                                                     *
*               cp/m-68k atari rbp bios                               *
*               basic input/output subsystem                          *
*               copyright 1984, atari corporation                     *
*               all rights reserved.                                  *
*               atari confidential                                    *
*                                                                     *
***********************************************************************
***********************************************************************
*                                                                     *
*       convert ikbd real-time clock format to jdos format            *
*                                                                     *
***********************************************************************

jdostime
        lea     $0,a5           ;address pointer to address base
        lea     clkrec(a5),a0
        bsr     bcdbin
        subi.b  #80,d0          ;adjust so that 1980 => 0 for time base
        move.b  d0,d2
        asl.l   #4,d2

        bsr     bcdbin
        add.b   d0,d2
        asl.l   #5,d2

        bsr     bcdbin
        add.b   d0,d2
        asl.l   #5,d2

        bsr     bcdbin
        add.b   d0,d2
        asl.l   #6,d2

        bsr     bcdbin
        add.b   d0,d2
        asl.l   #5,d2
        bsr     bcdbin
        lsr.b   d0              ;adjust to provide two second increments...
        add.b   d0,d2           ;...another @!#%@#$% kludge, thank you !
        move.l  d2,datetime(a5)
        move.b  #$0,newtod(a5)  ;clear handshaking flag
        rts

*************************************************************************
*                                                                       *
*               get time of day                                         *
*                                                                       *
*       entry:                                                          *
*                                                                       *
*       long    gettime()                                               *
*                                                                       *
*************************************************************************

        .globl  gettime

gettime
        move.b  #$-1,newtod(a5) ;set handshaking flag
        move.b  #gettod,d1      ;send get time of day command
        bsr     ikbdput
gtod1   tst.b   newtod(a5)              ;see if the new time of day is in yet..
        bne.b   gtod1
        move.l  datetime(a5),d0
        rts

*************************************************************************
*                                                                       *
*               set time of day                                         *
*                                                                       *
*       entry:                                                          *
*                                                                       *
*       void    settime(newtime)                                        *
*       long    newtime                                                 *
*                                                                       *
*************************************************************************

        .globl  settime

settime
        move.l  4(sp),newtime(a5)

*************************************************************************
*                                                                       *
*       convert jdos format to ikbd real-time clock format              *
*                                                                       *
*************************************************************************

        .globl  ikbdtime

ikbdtime
        lea     oclkrec+clkdex,a0       ;point to end of output clock buffer
        move.l  newtime(a5),d2  ;get time to convert
        move.b  d2,d0           ;make a copy for conversion routine
        andi.b  #%00011111,d0   ;mask off for pertinent information
        asl.b   d0              ;correct for the two second kludge
        bsr.b   binbcd          ;convert
        lsr.l   #5,d2           ;shift to next information field

        move.b  d2,d0           ;make a copy for conversion routine
        andi.b  #%00111111,d0   ;mask off for pertinent information
        bsr.b   binbcd          ;convert
        lsr.l   #6,d2           ;shift to next information field

        move.b  d2,d0           ;make a copy for conversion routine
        andi.b  #%00011111,d0   ;mask off for pertinent information
        bsr.b   binbcd          ;convert
        lsr.l   #5,d2           ;shift to next information field

        move.b  d2,d0           ;make a copy for conversion routine
        andi.b  #%00011111,d0   ;mask off for pertinent information
        bsr.b   binbcd          ;convert
        lsr.l   #5,d2           ;shift to next information field

        move.b  d2,d0           ;make a copy for conversion routine
        andi.b  #%00001111,d0   ;mask off for pertinent information
        bsr.b   binbcd          ;convert
        lsr.l   #4,d2           ;shift to next information field

        move.b  d2,d0           ;make a copy for conversion routine
        andi.b  #%01111111,d0   ;mask off for pertinent information
        bsr.b   binbcd          ;convert
        addi.b  #$80,(a0)       ;re-correct for ikbd format from jdos kludge

        move.b  #settod,d1      ;send set time-of-day command to ikbd
        bsr     ikbdput         ;use "inner circle" entry point!
        moveq   #clkdex-1,d3    ;prepare to send new parameters
        lea     oclkrec,a2      ;point to parameter list to be sent
        bsr     ikbdstr         ;again, use an "inner circle" entry point!
        move.b  #gettod,d1      ;send get time-of-day command to ikbd
        bsr     ikbdput         ;use "inner circle" entry point!
        rts

*************************************************************************
*                                                                       *
*               convert a byte from binary to bcd format                *
*                                                                       *
*       entry:  d0.l  - value                                           *
*                                                                       *
*************************************************************************

        .globl  binbcd

binbcd
        moveq   #0,d1
        moveq   #10,d3
bin2    sub.b   d3,d0
        bmi.b   bin1
        addq.b  #1,d1
        bra.b   bin2
```

```
bin1    addi.b  #10,d0
        asl.b   #4,d1
        add.b   d1,d0
        move.b  d0,-(a0)        ;transfer to output clock buffer
        rts

*********************************************************************
*                                                                   *
*               convert a byte from bcd format to binary            *
*                                                                   *
*       entry:  a0.1  - pointer to byte                             *
*                                                                   *
*********************************************************************

        .globl  bcdbin

bcdbin
        moveq   #$0,d0
        move.b  (a0),d0         ;get bcd byte
        lsr.b   #$4,d0          ;dump low nibble
        lsl.b   d0              ;generate (y1 shl 1)
        move.b  d0,d1           ;copy (y1 shl 1)
        asl.b   #2,d0           ;generate (y1 shl 3)
        add.b   d1,d0           ;generate (y1 shl 3) + (y1 shl 1)
        move.b  (a0)+,d1        ;grab bcd again for low nibble
        andi.w  #$f,d1          ;mask off for low nibble
        add.w   d1,d0           ;generate completed binary version of bcd byte
        rts

*********************************************************************
*                                                                   *
*               midi output status                                  *
*                                                                   *
*       entry:                                                      *
*                                                                   *
*       word    midiost()                                           *
*                                                                   *
*       returns true/okay to send = -1,  false/not ready = 0        *
*                                                                   *
*********************************************************************

        .globl  midiost

midiost
        moveq   #$-1,d0         ;pre-set to true
        move.b  comstat+midi,d2 ;grab midi status
        btst.l  #$1,d2
        bne.b   midiox          ;status okay to send
        moveq   #$0,d0          ;status not okay
midiox  rts

*********************************************************************
*                                                                   *
*               write char to midi port                             *
*                                                                   *
*       entry:                                                      *
*                                                                       *
*       void    midiwc(chr)                                             *
*       word    chr                                                     *
*                                                                       *
*************************************************************************

        .globl  midiwc

midiwc  move.w  6(sp),d1
midiput lea     midi,a1         ;point to midi register base
midput1 move.b  comstat(a1),d2  ;grab midi status
        btst.l  #$1,d2
        beq.b   midput1
        move.b  d1,iodata(a1)
        rts                     ;done for now


*************************************************************************
*                                                                       *
*               put string to midi routine                              *
*                                                                       *
*       entry:                                                          *
*                                                                       *
*       void    midiws(size,ptr)                                        *
*       word    size                                                    *
*       long    ptr                                                     *
*                                                                       *
*************************************************************************

        .globl  midiws

midiws  moveq   #$0,d3
        move.w  4(sp),d3        ;get size of string buffer - 1
        move.l  6(sp),a2        ;get string address
midp1   move.b  (a2)+,d1
        bsr.b   midiput
        dbra    d3,midp1
        rts

*************************************************************************
*                                                                       *
*               get midi receiver buffer status                         *
*                                                                       *
*       entry:                                                          *
*                                                                       *
*       word    midstat()                                               *
*                                                                       *
*       -1 signifies true/okay  0 - signifies false/no characters       *
*                                                                       *
*************************************************************************

        .globl  midstat

midstat lea     mbufrec(a5),a0  ;point to midi i/o bufrec
        lea     midi,a1         ;point to midi register base
        moveq   #$-1,d0         ;set result to true
        lea     ibufhead(a0),a2
        lea     ibuftail(a0),a3
        cmpm.w  (a3)+,(a2)+     ;atomic buffer empty test
        bne.b   midist1         ;branch if not, assume d0 is "clr.w"'ed
        moveq   #$0,d0          ;set result to false
midist1 rts


**********************************************************************
*                                                                    *
*               getchar routine for midi port                        *
*                                                                    *
*       this routine transfers characters from a input queue that is *
*       filled by an automatic interrupt routine.  the interrupt     *
*       routine handles the actual transfer of the character from the *
*       i/o port.                                                    *
*                                                                    *
*       entry:                                                       *
*                                                                    *
*       long    midin()                                              *
*                                                                    *
*       long data returned represents upper three bytes of time stamp *
*       and least significant byte as data                           *
*                                                                    *
**********************************************************************

        .globl  midin

midin

*       assume that a0/a1 are inited by the midstat call for the rest of
*       this routine.

        bsr.b   midstat         ;see if key pressed
        tst.w   d0
        beq.b   midin           ;wait until byte comes in
        move    sr,-(sp)        ;protect this upcoming test
        ori     #$700,sr
        move.w  ibufhead(a0),d1 ;get current head pointer offset from buffer
        cmp.w   ibuftail(a0),d1 ;head=tail?
        beq.b   mwi2            ;yes

*       check for wrap of pointer

        addq.w  #1,d1           ;i=h+1
        cmp.w   ibufsiz(a0),d1  ;? i>= current bufsiz?
        bcs.b   mwi1            ;no...
        moveq   #$0,d1          ;wrap pointer
mwi1    move.l  ibufptr(a0),a1  ;get base address of buffer
        move.b  0(a1,d1),d0     ;get character
        move.w  d1,ibufhead(a0) ;store new head pointer to buffer record
mwi2    move    (sp)+,sr
        rts

**********************************************************************
*                                                                    *
*               parallel i/o port service routine                            *
*                                                                            *
*       this set of routines is for general parallel i/o                     *
*                                                                            *
*       entry to listout                                                     *
*                                                                            *
*       entry to listin                                                      *
*                                                                            *
*       exit from listin                                                     *
*                                                                            *
******************************************************************************

        .globl  _lstout

_lstout
        move.l  _hz_200(a5),d2  ; d2 = hz_200 - prt_to
        sub.l   prt_to(a5),d2   ; (compute time since last timeout)
        cmpi.l  #5*200,d2       ; do "fake" timeout if we timed out within
        bcs.b   lperr           ; the last five seconds

        move.l  _hz_200(a5),d2  ; d2 = starting time for this char
pt0     bsr.b   _lstostat       ;go get parallel port status
        tst.w   d0              ;...and check for high (busy)
        bne.b   pt1             ; port is ready -- print the char

        move.l  _hz_200(a5),d3  ; d3 = hz_200 - d2
        sub.l   d2,d3
        cmpi.l  #30*200,d3      ; check for 30 second delta
        blt.b   pt0             ; continue if no timeout

lperr   moveq   #$0,d0          ; return value of 0 indicates timeout
        move.l  _hz_200(a5),prt_to(a5)  ; record time of last timeout
        rts

pt1     move.w  sr,d3           ;save status register
        ori.w   #$700,sr        ;protect upcoming switching of the port setting
        moveq   #mixer,d1       ;get current io enable register contents
        bsr     gientry
        ori.b   #$80,d0         ;set port b for output
        moveq   #mixer+$80,d1   ;set to write to io enable
        bsr     gientry
        move.w  d3,sr           ;restore status register

        move.w  6(sp),d0        ;retrieve byte to be sent and...
        moveq   #portb+$80,d1   ;write out byte to parallel port
        bsr     gientry

        bsr.b   strobeon
        bsr.b   strobeoff
        moveq   #$-1,d0         ;set d0=-1 for good transfer status
lexit   rts

strobeoff
        moveq   #%00100000,d2   ;set strobe off
        bra     onbit           ;go set it!!
```

```
strobeon
        moveq   #%11011111,d2   ;set strobe on
        bra     offbit          ;set strobe now...


        .globl  _lstin

_lstin  moveq   #mixer,d1       ;get current io enable register contents
        bsr     gientry
        andi.b  #$7f,d0         ;set port b for input
        moveq   #mixer+$80,d1   ;set to write to io enable
        bsr     gientry

        bsr.b   strobeoff       ;busy off!
lstibusy
        bsr.b   _lstostat       ;go get parallel port status
        tst.w   d0              ;...and check for high (busy)
        bne.b   lstibusy        ;loop till high...
        bsr.b   strobeon
        moveq   #portb,d1       ;init to use gientry routine to read
        bra     gientry         ;now get the byte from the parallel port
*                               ;d0.l contains the byte of data from the port
*       the 'bra' is implied rts from this routine

************************************************************************
*                                                                      *
*               parallel port status routine                           *
*                                                                      *
************************************************************************
        .globl  _lstostat

_lstostat
        lea     mfp,a0          ;point to mfp register base
        moveq   #$-1,d0         ;pre-init to true (parallel port ready)
        btst.b  #$0,gpip(a0)
        beq.b   lst1
        moveq   #$0,d0          ;parallel port busy
lst1    rts


************************************************************************
*                                                                      *
*               auxillary port input status routine                    *
*                                                                      *
************************************************************************

        .globl  auxistat

auxistat
        lea     rbufrec(a5),a0  ;point to rs-232 buffer record
        moveq   #$-1,d0         ;set result to true
        lea     ibufhead(a0),a2
        lea     ibuftail(a0),a3
        cmpm.w  (a3)+,(a2)+     ;atomic buffer empty test
        bne.b   auxist1
        moveq   #$0,d0          ;set result to false
```

```
auxistl rts

**********************************************************************
*                                                                    *
*               auxillary input routine                              *
*                                                                    *
**********************************************************************

        .globl  auxin

auxin   bsr.b   auxistat        ;see if key pressed
        tst.w   d0
        beq.b   auxin           ;wait until key pressed
        bsr     rs232get
        andi.w  #$ff,d0         ;clear out the high byte
        rts

**********************************************************************
*                                                                    *
*               auxillary port output status routine                 *
*                                                                    *
**********************************************************************

        .globl  _auxostat

_auxostat
        lea     rbufrec(a5),a0  ;point to rs-232 buffer record
        moveq   #$-1,d0         ;set result to true
        move.w  obuftail(a0),d2 ;get current tail pointer offset from buffer
        bsr     wrapout         ;check for wrap of pointer
        cmp.w   obufhead(a0),d2 ;head=tail?
        bne.b   auxostl         ;no...there is buffer space left!
        moveq   #$0,d0          ;set result to false
auxostl rts

**********************************************************************
*                                                                    *
*               auxillary output routine                             *
*                                                                    *
**********************************************************************

        .globl  _auxout

_auxout move.w  6(sp),d1        ;get data
        bsr     rs232put        ;exit via rs-232 output routine
        bcs.b   _auxout
        rts

**********************************************************************
*                                                                    *
*               ikbd output status                                   *
*                                                                    *
*       entry:                                                       *
*                                                                    *
*       word    ikbdost()                                            *
*                                                                    *
*       returns true/okay to send = -1,  false/not ready = 0           *
*                                                                      *
************************************************************************

        .globl  ikbdost

ikbdost
        moveq   #$-1,d0         ;pre-set to true
        move.b  comstat+keyboard,d2     ;grab ikbd status
        btst.l  #$1,d2
        bne.b   ikbdox          ;status okay to send
        moveq   #$0,d0          ;status not okay
ikbdox  rts

************************************************************************
*                                                                      *
*               write char to ikbd port                                *
*                                                                      *
*       entry:                                                         *
*                                                                      *
*       void    ikbdwc(chr)                                            *
*       word    chr                                                    *
*                                                                      *
*                                                                      *
************************************************************************

        .globl  ikbdwc

ikbdwc  move.w  6(sp),d1
ikbdput lea     keyboard,a1     ;point to ikbd register base
ikput1  move.b  comstat(a1),d2  ;grab keyboard status
        btst.l  #$1,d2
        beq.b   ikput1
        move.b  d1,iodata(a1)
        rts                     ;done for now


************************************************************************
*                                                                      *
*               put string to ikbd routine                             *
*                                                                      *
*       entry:                                                         *
*                                                                      *
*       void    ikbdws(size,ptr)                                       *
*       word    size                                                   *
*       long    ptr                                                    *
*                                                                      *
************************************************************************

        .globl  ikbdws

ikbdws  moveq   #$0,d3
        move.w  4(sp),d3
        move.l  6(sp),a2
ikbdstr move.b  (a2)+,d1
        bsr.b   ikbdput
```

```
          dbra     d3,ikbdstr
          rts

          .globl   constat

constat
          lea      kbufrec(a5),a0  ;point to ikbd buffer record
          moveq    #$-1,d0         ;set result to true
          lea      ibufhead(a0),a2
          lea      ibuftail(a0),a3
          cmpm.w   (a3)+,(a2)+     ;atomic buffer empty test
          bne.b    const1          ;branch if not, assume d0 is "clr.w"'ed
          moveq    #$0,d0          ;set result to false
const1    rts

          .globl   conin

conin     bsr.b    constat         ;see if key pressed
          tst.w    d0
          beq.b    conin           ;wait until key pressed
          move     sr,-(sp)        ;protect this upcoming test
          ori      #$700,sr
          move.w   ibufhead(a0),d1 ;get current head pointer offset from buffer
          cmp.w    ibuftail(a0),d1 ;head=tail?
          beq.b    cwi2            ;yes

*         check for wrap of pointer

          addq.w   #2,d1           ;i=h+2
          cmp.w    ibufsiz(a0),d1  ;? i>= current bufsiz?
          bcs.b    cwi1            ;no...
          moveq    #$0,d1          ;wrap pointer
cwi1      move.l   ibufptr(a0),a1  ;get base address of buffer
          moveq    #$0,d0          ;clear out for jdos format
          move.w   0(a1,d1),d0     ;get character
          move.w   d1,ibufhead(a0) ;store new head pointer to buffer record
          lsl.l    #$8,d0          ;shift the scancode only to the low byte
          lsr.w    #$8,d0          ;high word location for jdos
cwi2      move     (sp)+,sr
          rts


          .globl   conoutst

conoutst
          moveq    #-1,d0
          rts                      ;jdos requirement

          .globl   ringbel

ringbel
          btst.b   #$2,conterm(a5)
          beq.b    rgbel
          move.l   #bellsnd,cursnd(a5)
          move.b   #0,timer(a5)
rgbel     rts
```

```
*************************************************************************
*                                                                       *
*       end of gemdos bios portion                                      *
*                                                                       *
*       device driver and auxillary routines follow                     *
*                                                                       *
*************************************************************************

        ifeq    COUNTRY-USA

keytran:
        dc.b    $00,$1b,'1','2','3','4','5','6'
        dc.b    '7','8','9','0','-','=',$08,$09
        dc.b    'q','w','e','r','t','y','u','i'
        dc.b    'o','p','[',']',$0D,$00,'a','s'
        dc.b    'd','f','g','h','j','k','l',';'
        dc.b    $27,'`',$00,'\','z','x','c','v'
        dc.b    'b','n','m',',','.','/',$00,$00
        dc.b    $00,$20,$00,$00,$00,$00,$00,$00

        dc.b    $00,$00,$00,$00,$00,$00,$00,$00
        dc.b    $00,$00,'-',$00,$00,$00,'+',$00
        dc.b    $00,$00,$00,$7f,$00,$00,$00,$00
        dc.b    $00,$00,$00,$00,$00,$00,$00,$00
        dc.b    $00,$00,$00,'(',')','/','*','7'
        dc.b    '8','9','4','5','6','1','2','3'
        dc.b    '0','.',$0D,$00,$00,$00,$00,$00
        dc.b    $00,$00,$00,$00,$00,$00,$00,$00

keyshif:
        dc.b    $00,$1b,'!','@','#','$','%','^'
        dc.b    '&','*','(',')','_','+',$08,$09
        dc.b    'Q','W','E','R','T','Y','U','I'
        dc.b    'O','P','{','}',$0D,$00,'A','S'
        dc.b    'D','F','G','H','J','K','L',':'
        dc.b    '"','~',$00,'|','Z','X','C','V'
        dc.b    'B','N','M','<','>','?',$00,$00
        dc.b    $00,$20,$00,$00,$00,$00,$00,$00

        dc.b    $00,$00,$00,$00,$00,$00,$00,$37
        dc.b    $38,$00,'-',$34,$00,$36,'+',$00
        dc.b    $32,$00,$30,$7f,$00,$00,$00,$00
        dc.b    $00,$00,$00,$00,$00,$00,$00,$00
        dc.b    $00,$00,$00,'(',')','/','*','7'
        dc.b    '8','9','4','5','6','1','2','3'
        dc.b    '0','.',$0D,$00,$00,$00,$00,$00
        dc.b    $00,$00,$00,$00,$00,$00,$00,$00

keycl:
        dc.b    $00,$1b,'1','2','3','4','5','6'
        dc.b    '7','8','9','0','-','=',$08,$09
        dc.b    'Q','W','E','R','T','Y','U','I'
        dc.b    'O','P','[',']',$0D,$00,'A','S'
        dc.b    'D','F','G','H','J','K','L',';'
        dc.b    $27,'`',$00,'\','Z','X','C','V'
        dc.b    'B','N','M',',','.','/',$00,$00
        dc.b    $00,$20,$00,$00,$00,$00,$00,$00

        dc.b    $00,$00,$00,$00,$00,$00,$00,$00
        dc.b    $00,$00,'-',$00,$00,$00,'+',$00
        dc.b    $00,$00,$00,$7f,$00,$00,$00,$00
        dc.b    $00,$00,$00,$00,$00,$00,$00,$00
        dc.b    $00,$00,$00,'(',')','/','*','7'
        dc.b    '8','9','4','5','6','1','2','3'
        dc.b    '0','.',$0D,$00,$00,$00,$00,$00
        dc.b    $00,$00,$00,$00,$00,$00,$00,$00

        .endc


        ifeq    COUNTRY-UK

keytran:
        dc.b    $00,$1b,'1','2','3','4','5','6'
        dc.b    '7','8','9','0','-','=',$08,$09
        dc.b    'q','w','e','r','t','y','u','i'
        dc.b    'o','p','[',']',$0D,$00,'a','s'
        dc.b    'd','f','g','h','j','k','l',';'
        dc.b    $27,'`',$00,'#','z','x','c','v'
        dc.b    'b','n','m',',','.','/',$00,$00
        dc.b    $00,$20,$00,$00,$00,$00,$00,$00

        dc.b    $00,$00,$00,$00,$00,$00,$00,$00
        dc.b    $00,$00,'-',$00,$00,$00,'+',$00
        dc.b    $00,$00,$00,$7f,$00,$00,$00,$00
        dc.b    $00,$00,$00,$00,$00,$00,$00,$00
        dc.b    '\',$00,$00,'(',')','/','*','7'
        dc.b    '8','9','4','5','6','1','2','3'
        dc.b    '0','.',$0D,$00,$00,$00,$00,$00
        dc.b    $00,$00,$00,$00,$00,$00,$00,$00

keyshif:
        dc.b    $00,$1b,'!','"',$9c,'$','%','^'
        dc.b    '&','*','(',')','_','+',$08,$09
        dc.b    'Q','W','E','R','T','Y','U','I'
        dc.b    'O','P','{','}',$0D,$00,'A','S'
        dc.b    'D','F','G','H','J','K','L',':'
        dc.b    '@',$ff,$00,'~','Z','X','C','V'
        dc.b    'B','N','M','<','>','?',$00,$00
        dc.b    $00,$20,$00,$00,$00,$00,$00,$00

        dc.b    $00,$00,$00,$00,$00,$00,$00,$37
        dc.b    $38,$00,'-',$34,$00,$36,'+',$00
        dc.b    $32,$00,$30,$7f,$00,$00,$00,$00
        dc.b    $00,$00,$00,$00,$00,$00,$00,$00
        dc.b    '|',$00,$00,'(',')','/','*','7'
        dc.b    '8','9','4','5','6','1','2','3'
        dc.b    '0','.',$0D,$00,$00,$00,$00,$00
        dc.b    $00,$00,$00,$00,$00,$00,$00,$00
```

```
keycl:
            dc.b    $00,$1b,'1','2','3','4','5','6'
            dc.b    '7','8','9','0','-','=',$08,$09
            dc.b    'Q','W','E','R','T','Y','U','I'
            dc.b    'O','P','[',']',$0d,$00,'A','S'
            dc.b    'D','F','G','H','J','K','L',';'
            dc.b    $27,'`',$00,'#','Z','X','C','V'
            dc.b    'B','N','M',',','.','/',$00,$00
            dc.b    $00,$20,$00,$00,$00,$00,$00,$00

            dc.b    $00,$00,$00,$00,$00,$00,$00,$00
            dc.b    $00,$00,'-',$00,$00,$00,'+',$00
            dc.b    $00,$00,$00,$7f,$00,$00,$00,$00
            dc.b    $00,$00,$00,$00,$00,$00,$00,$00
            dc.b    '\',$00,$00,'(',')','/','*','7'
            dc.b    '8','9','4','5','6','1','2','3'
            dc.b    '0','.',$0D,$00,$00,$00,$00,$00
            dc.b    $00,$00,$00,$00,$00,$00,$00,$00


            .endc


            ifeq    COUNTRY-GERMANY

keytran:
            dc.b    $00,$1b,'1','2','3','4','5','6'
            dc.b    '7','8','9','0',$9e,$27,$08,$09
            dc.b    'q','w','e','r','t','z','u','i'
            dc.b    'o','p',$81,'+',$0D,$00,'a','s'
            dc.b    'd','f','g','h','j','k','l',$94
            dc.b    $84,'#',$00,'~','y','x','c','v'
            dc.b    'b','n','m',',','.','-',$00,$00
            dc.b    $00,$20,$00,$00,$00,$00,$00,$00

            dc.b    $00,$00,$00,$00,$00,$00,$00,$00
            dc.b    $00,$00,'-',$00,$00,$00,'+',$00
            dc.b    $00,$00,$00,$7f,$00,$00,$00,$00
            dc.b    $00,$00,$00,$00,$00,$00,$00,$00
            dc.b    '<',$00,$00,'(',')','/','*','7'
            dc.b    '8','9','4','5','6','1','2','3'
            dc.b    '0','.',$0D,$00,$00,$00,$00,$00
            dc.b    $00,$00,$00,$00,$00,$00,$00,$00

keyshif:
            dc.b    $00,$1b,'!','"',$dd,'$','%','&'
            dc.b    '/','(',')','=','?','`',$08,$09
            dc.b    'Q','W','E','R','T','Z','U','I'
            dc.b    'O','P',$9a,'*',$0D,$00,'A','S'
            dc.b    'D','F','G','H','J','K','L',$99
            dc.b    $8e,'^',$00,'|','Y','X','C','V'
            dc.b    'B','N','M',';',':','_',$00,$00
            dc.b    $00,$20,$00,$00,$00,$00,$00,$00

            dc.b    $00,$00,$00,$00,$00,$00,$00,$37
            dc.b    $38,$00,'-',$34,$00,$36,'+',$00
```

```
          dc.b    $32,$00,$30,$7f,$00,$00,$00,$00
          dc.b    $00,$00,$00,$00,$00,$00,$00,$00
          dc.b    '>',$00,$00,'(',')','/','*','7'
          dc.b    '8','9','4','5','6','1','2','3'
          dc.b    '0','.',$0D,$00,$00,$00,$00,$00
          dc.b    $00,$00,$00,$00,$00,$00,$00,$00

keycl:
          dc.b    $00,$1b,'1','2','3','4','5','6'
          dc.b    '7','8','9','0',$9e,$27,$08,$09
          dc.b    'Q','W','E','R','T','Z','U','I'
          dc.b    'O','P',$9a,'+',$0D,$00,'A','S'
          dc.b    'D','F','G','H','J','K','L',$99
          dc.b    $8e,'#',$00,'~','Y','X','C','V'
          dc.b    'B','N','M',',','.','-',$00,$00
          dc.b    $00,$20,$00,$00,$00,$00,$00,$00

          dc.b    $00,$00,$00,$00,$00,$00,$00,$00
          dc.b    $00,$00,'-',$00,$00,$00,'+',$00
          dc.b    $00,$00,$00,$7f,$00,$00,$00,$00
          dc.b    $00,$00,$00,$00,$00,$00,$00,$00
          dc.b    '<',$00,$00,'(',')','/','*','7'
          dc.b    '8','9','4','5','6','1','2','3'
          dc.b    '0','.',$0D,$00,$00,$00,$00,$00
          dc.b    $00,$00,$00,$00,$00,$00,$00,$00

          .endc


          ifeq    COUNTRY-FRANCE

keytran:
          dc.b    $00,$1b,'&',$82,'"',$27,'(',$dd
          dc.b    $8a,'!',$80,$85,')','-',$08,$09
          dc.b    'a','z','e','r','t','y','u','i'
          dc.b    'o','p','^','$',$0D,$00,'q','s'
          dc.b    'd','f','g','h','j','k','l','m'
          dc.b    $97,'`',$00,'#','w','x','c','v'
          dc.b    'b','n',',',';',':','=',$00,$00
          dc.b    $00,$20,$00,$00,$00,$00,$00,$00

          dc.b    $00,$00,$00,$00,$00,$00,$00,$00
          dc.b    $00,$00,'-',$00,$00,$00,'+',$00
          dc.b    $00,$00,$00,$7f,$00,$00,$00,$00
          dc.b    $00,$00,$00,$00,$00,$00,$00,$00
          dc.b    '<',$00,$00,'(',')','/','*','7'
          dc.b    '8','9','4','5','6','1','2','3'
          dc.b    '0','.',$0D,$00,$00,$00,$00,$00
          dc.b    $00,$00,$00,$00,$00,$00,$00,$00

keyshif:
          dc.b    $00,$1b,'1','2','3','4','5','6'
          dc.b    '7','8','9','0',$f8,$ff,$08,$09
          dc.b    'A','Z','E','R','T','Y','U','I'
          dc.b    'O','P',$b9,'*',$0D,$00,'Q','S'
          dc.b    'D','F','G','H','J','K','L','M'
```

```
        dc.b    '%',$9c,$00,'!','W','X','C','V'
        dc.b    'B','N','?','.','/','+',$00,$00
        dc.b    $00,$20,$00,$00,$00,$00,$00,$00

        dc.b    $00,$00,$00,$00,$00,$00,$00,$37
        dc.b    $38,$00,'-',$34,$00,$36,'+',$00
        dc.b    $32,$00,$30,$7f,$00,$00,$00,$00
        dc.b    $00,$00,$00,$00,$00,$00,$00,$00
        dc.b    '>',$00,$00,'(',')','/','*','7'
        dc.b    '8','9','4','5','6','1','2','3'
        dc.b    '0','.',$0D,$00,$00,$00,$00,$00
        dc.b    $00,$00,$00,$00,$00,$00,$00,$00

keycl:
        dc.b    $00,$1b,'&',$82,'"',$27,'(',$dd
        dc.b    $8a,'!',$80,$85,')','-',$08,$09
        dc.b    'A','Z','E','R','T','Y','U','I'
        dc.b    'O','P','^','$',$0D,$00,'Q','S'
        dc.b    'D','F','G','H','J','K','L','M'
        dc.b    $97,'`',$00,'#','W','X','C','V'
        dc.b    'B','N',',',';',':','=',$00,$00
        dc.b    $00,$20,$00,$00,$00,$00,$00,$00

        dc.b    $00,$00,$00,$00,$00,$00,$00,$00
        dc.b    $00,$00,'-',$00,$00,$00,'+',$00
        dc.b    $00,$00,$00,$7f,$00,$00,$00,$00
        dc.b    $00,$00,$00,$00,$00,$00,$00,$00
        dc.b    '<',$00,$00,'(',')','/','*','7'
        dc.b    '8','9','4','5','6','1','2','3'
        dc.b    '0','.',$0D,$00,$00,$00,$00,$00
        dc.b    $00,$00,$00,$00,$00,$00,$00,$00

        .endc


        .even
        .page
        .text
***********************************************************************
*                                                                     *
*       routine to set up the general interrupt port registers        *
*               (gpip,are,ddr)                                        *
*                                                                     *
*       algorithm to set up the port                                  *
*                                                                     *
*       1. mask off all interrupts via the imrx registers;            *
*       2. clear all enable and pending bits in the ierx and iprx     *
*               registers;                                            *
*       3. check the interrupt in-service registers and loop till     *
*               clear;                                                *
*       4. init the aer register bits as desired (default = 11111111); *
*       5. init the ddr register bits as desired (default = 10000000); *
*       6. clear the gpip register;                                   *
*       7. enable all desired interrupt enable bits;                  *
*       8. mask on all desired interrupt mask bits;                   *
*                                                                     *
*                                                                         *
***************************************************************************

        .globl  initmfp

initmfp
        lea     mfp,a0          ;init mfp address pointer

        moveq   #$0,d0          ;init to zero for clearing mfp
        movep.l d0,gpip(a0)     ;clear gpip thru iera
        movep.l d0,ierb(a0)     ;clear ierb thru isrb
        movep.l d0,isrb(a0)     ;clear isrb thru vr

        move.b  #$48,vr(a0)     ;set mfp autovector and s-bit
*       move.b  #$4,aer(a0)     ;set cts to low to high transition

*       init the "c" timer

        move.w  #$1111,tc_rot(a5)       ;setup bitstream for /4 on timer c interr
        move.w  #20,_timr_ms(a5)        ;set timer calibration value

        moveq   #ctimer,d0      ;set to timer C
        moveq   #$50,d1         ;set to /64 for 200 hz tick
        move.w  #192,d2         ;set to 192
        bsr     setimer         ;setup timer and init interrupt vector.....

        lea     timercint,a2    ;point to the timer C interrupt routine...
        moveq   #$5,d0          ;point to the timer C interrupt number
        bsr     initint

*       init the "d" timer

        moveq   #dtimer,d0      ;select the d timer
        moveq   #c9600,d1       ;init for /4 for 9600 baud
        moveq   #d9600,d2       ;init for 9600 baud
*       moveq   #c1200,d1       ;init for /4 for 9600 baud
*       moveq   #d1200,d2       ;init for 9600 baud
        bsr     setimer         ;branch to our timer initialier...

*       now init the 3 rs232 chip registers

        move.l  #$00980101,d0
        movep.l d0,scr(a0)      ;inits scr,ucr,rsr,tsr

*       initialize the default rs-232 control line settings

        bsr     dtron
        bsr     rtson

*       initialize the rs-232 buffer record structure

        lea     rbufrec(a5),a0
        lea     rs232init,a1
        moveq   #rssize,d0
        bsr     lbmove          ;do block move and return
```

```
*       initialize the midi buffer record structure

        lea     mbufrec(a5),a0
        lea     minit,a1
        moveq   #mssize,d0
        bsr     lbmove              ;do block move and return

        move.l  #aciaexit,d0      ;init to ikbd and midi error handler address
        move.l  d0,vkbderr(a5)    ;init keyboard error handler address
        move.l  d0,vmiderr(a5)    ;init midi error handler address
        move.l  #sysmidi,midivec(a5)     ;point to system midi interrupt vector

*       init the midi acia next

        move.b  #rsetacia,comstat+midi   ;init the acia via master reset

* init the acia to divide by 16x clock, 8 bit data, 1 stop bit, no parity,
* rts low, transmitting interrupt disabled, receiving interrupt enabled

        move.b  #div16+protocol+rtsld+intron,comstat+midi

*       initialize the keyboard acia interrupt vector exception address

        move.b  #%00000111,conterm(a5)   ;enable keyclick,repeat key,bell functions

        move.l  #jdostime,clkintvec(a5)
        move.l  #genrts,d0        ;generalized rts for ikbd subsystems
        move.l  d0,statintvec(a5)
        move.l  d0,msintvec(a5) ;init user mouse interrupt adr to rts
        move.l  d0,joyintvec(a5)
*
*  Sound routine initialization - uses the pre-init'ed d0.l=0000 !!
*
*initsnd:
        moveq   #$0,d0                      ;init 'd0' to clear sound variables
        move.l  d0,cursnd(a5)               ;clear sound ptr
        move.b  d0,timer(a5)                ;clear delay timer
        move.b  d0,auxd(a5)                 ;clear temp value
        move.l  d0,prt_to(a5)               ;init printer timout to 0

        bsr     strobeoff                   ;init strobe to off (line high!)
        move.b  #tdelay1,cdelay1(a5)        ;init system default key repeat values
        move.b  #tdelay2,cdelay2(a5)

* within the mouse relative routine

*       initialize the ikbd buffer record structure

        lea     kbufrec(a5),a0
        lea     kinit,a1
        moveq   #kssize,d0
        bsr.b   lbmove              ;do block move and return

        bsr     bioskeys                    ;point key translation address to
*                                           ;the rom based translation tables
*       init the acia next

        move.b  #rsetacia,comstat+keyboard      ;init the acia via master re

* now that the vector is initialized, we can allow interrupts to occur!
* init the acia to divide by 64 clock, 8 bit data, 1 stop bit, no parity,
* rts low, transmitting interrupt disabled, receiving interrupt enabled

        move.b  #div64+protocol+rtsld+intron,comstat+keyboard

        move.l  #mfpvectr,a3    ;point to initializing array of exception vec's
        moveq   #$3,d1          ;init branch counter/index
st1     move.l  d1,d2
        move.l  d1,d0           ;load in interrupt # to setup
        addi.b  #$9,d0          ;add constant to point to proper mfp interrupt
        asl.l   #2,d2
        move.l  0(a3,d2),a2
        bsr     initint         ;go to service routine
        dbra    d1,st1
        lea     midikey(a5),a2
        moveq   #$6,d0          ;load in interrupt # to setup
        bsr     initint         ;go to service routine

        lea     ctsint(a5),a2   ;point to the CTS interrupt routine...
        moveq   #$2,d0          ;point to the CTS interrupt number
        bsr     initint

*                               ;initializing code which sets the enable
*                               ;and mask bits...
        movea.l #setikbd,a2
        moveq   #sizeikbd,d3
        bsr     ikbdstr         ;init ikbd from 'setikbd' data

genrts  rts

lbmove  move.b  (a1)+,(a0)+
        dbra    d0,lbmove
        rts                     ;and return home

setikbd dc.b    $80,$01,$12,$1a ;reset keyboard,disable mouse,disable joysticks

sizeikbd        equ     *-setikbd-1

kinit
        dc.l    kibuffer
        dc.w    kinsize
        dc.w    0
        dc.w    0
        dc.w    kinsize/4
        dc.w    kinsize*3/4

kssize  equ     *-kinit-1

minit
        dc.l    mibuffer
        dc.w    minsize
        dc.w    0
        dc.w    0
        dc.w    minsize/4
        dc.w    minsize*3/4

mssize  equ     *-minit-1

        .even

rs232init
        dc.l    ribuffer        ;ibufptr
        dc.w    rinsize         ;ibufsiz
        dc.w    0               ;ibufhead
        dc.w    0               ;ibuftail
        dc.w    rinsize/4       ;ibuflow
        dc.w    rinsize*3/4     ;ibufhigh

        dc.l    robuffer        ;obufptr
        dc.w    routsize        ;obufsiz
        dc.w    0               ;obufhead
        dc.w    0               ;obuftail
        dc.w    routsize/4      ;obuflow
        dc.w    routsize*3/4    ;obufhigh
        dc.b    0               ;rsrbyte
        dc.b    0               ;tsrbyte
        dc.b    0               ;rxoff
        dc.b    0               ;txoff
        dc.b    1               ;rsmode -- xon/xoff mode
*       dc.b    2               ;rsmode -- CTS/RTS/DTR mode
        dc.b    0               ;rsmode filler

rssize  equ     *-rs232init-1

        .even


mfpvectr

*       array of exception vector addresses for the above interrupts, including
*       dummy vectors that point to "rte's".

        dc.l    txerror
        dc.l    txrint
        dc.l    rxerror
        dc.l    rcvrint


        .page
        .text
*************************************************************************
*                                                                       *
*                       routine to setup a timer                        *
*                                                                       *
*       algorithm to init a timer                                       *
*                                                                       *
*       1. determine which timer and set d0.b = to timer's index value  *
*          as shown below;                                              *
*       2. disable the associated interrupt;                            *
*       3. disable the timer itself via it's timer control register;    *
*       4. initialize the timer's data register                         *
*       5. repeat step #4 urtil the data register's contents are        *
*          verified, per the errata sheet to the 68901 description;     *
*       6. turn on the timer by using the value that you previously     *
*          stored in d1;                                                *
*                                                                       *
*       note:   the interrupt vector for the associated timer           *
*               is not set in this routine, so it is the user's         *
*               responsiblity to set it if so desired!                  *
*                                                                       *
*                                                                       *
*       registers used:         d0-d3/a0-a3                             *
*       registers saved:        d0-d3/a0-a3                             *
*       entry:                                                          *
*               d0.l - timer to be set                                  *
*                       0 - timer a                                     *
*                       1 - timer b                                     *
*                       2 - timer c                                     *
*                       3 - timer d                                     *
*               d1.b - timer's new control setting                      *
*               d2.b - timer's data register data                       *
*                                                                       *
*       exit:   no values to pass                                       *
*                                                                       *
*               d3   - used and abused by call to mskreg routine        *
*               a0.l - set to mfp register base                         *
*               a1.l - temporary location for a3                        *
*               a2.l - used to pass table address to mskreg routine     *
*               a3.l - used to pass table address to mskreg routine     *
*                                                                       *
*************************************************************************

        .globl  setimer

setimer:
        movem.l d0-d4/a0-a3,-(sp)       ;save all registers to be messed with!!
        move.l  #mfp,a0         ;set mfp chip address pointer

        move.l  #imrt,a3        ;mask off the timer's interrupt maskable bit
        move.l  #imrmt,a2
        bsr.b   mskreg

        move.l  #iert,a3        ;mask off the timer's interrupt enable bit
        move.l  #iermt,a2
        bsr.b   mskreg

        move.l  #iprt,a3        ;mask off the timer's interrupt pending bit
        move.l  #iprmt,a2
        bsr.b   mskreg

        move.l  #isrt,a3        ;mask off the timer's interrupt inservice bit
        move.l  #isrmt,a2
        bsr.b   mskreg
        move.l  #tcrtab,a3          ;mask off the timer's control bits
        move.l  #tcrmsk,a2
        bsr.b   mskreg

        exg     a3,a1               ;save address pointer for restoring control

        lea     tdrtab,a3           ;initialize the timer data register
        moveq   #$0,d3              ;to prevent false effective address generation
        move.b  0(a3,d0),d3
verify  move.b  d2,0(a0,d3)
        cmp.b   0(a0,d3),d2
        bne.b   verify

        exg     a3,a1               ;grab that register address back
        or.b    d1,(a3)             ;mask the timer control register value

        movem.l (sp)+,d0-d4/a0-a3       ;restore all registers that were saved
        rts

***********************************************************************
*               generalize mask register bit(s) routine               *
*                                                                     *
*       entry                                                         *
*       static  d0 - contains the timer #                             *
*               d3 - used and abused                                  *
*               d4 - used and abused                                  *
*       static  a0 - mfp register base                                *
*               a3 - points to table of similar timer registers       *
*       static  a2 - points to table of similar timer data values     *
***********************************************************************

mskreg
        bsr.b   getmask
        move.b  (a2),d3             ;grab mask now
        and.b   d3,(a3)             ;and have masked off the desired bit(s)
        rts

getmask moveq   #$0,d3              ;to prevent false effective address generation
        adda    d0,a3               ;have got pointer to mfp register now
        move.b  (a3),d3             ;now have the address offset to mfp
        add.l   a0,d3
        movea.l d3,a3               ;now have address pointing to desired mfp reg.
*                                   ;now we get the mask to turn off interrupt
        adda    d0,a2               ;have got pointer to mask now
        rts

iert    dc.b    $6,$6,$8,$8
iprt    dc.b    $A,$A,$C,$C
isrt    dc.b    $E,$E,$10,$10
imrt    dc.b    $12,$12,$14,$14

iermt   dc.b    $df,$fe,$df,$ef
imrmt   equ     iermt
iprmt   equ     iermt
isrmt   equ     iermt
```

```
tcrtab  dc.b    $18,$1a,$1c,$1c
tcrmsk  dc.b    $0,$0,$8f,$f8
tdrtab  dc.b    $1e,$20,$22,$24

        .even

**************************************************************************
*                                                                        *
*       initialize mfp interrupt via GEMDOS                              *
*                                                                        *
*       entry                                                            *
*                                                                        *
*       void    mfpint(numint,intvec)                                    *
*       word    numint                                                   *
*       long    intvec                                                   *
*                                                                        *
*                                                                        *
**************************************************************************

        .globl  mfpint

mfpint
        move.w  4(sp),d0
        move.l  6(sp),a0
        andi.l  #$f,d0          ;to ensure masking of 0-$f

**************************************************************************
*                                                                        *
*       routine to init an mfp associated interrupt vector               *
*                                                                        *
*       algorithm                                                        *
*                                                                        *
*       1. block the interrupt via it's mask bit;                        *
*       2. disable the interrupt's enable and pending bits;              *
*       3. check the interrupt's in-service register and loop till       *
*          clear;                                                        *
*       4. init the interrupt's associated vector;                       *
*       5. set the interrupt's enable bit;                               *
*       6. set the interrupt's mask bit;                                 *
*                                                                        *
*       entry                                                            *
*               d0 - contains interrupt # to affect                      *
*               a2 - contains new vector address                         *
**************************************************************************

initint
        movem.l d0-d2/a0-a2,-(sp)       ;save affected registers
        bsr.b   disint          ;disable the interrupts
        move.l  d0,d2           ;get a copy so as to determine where to...
        asl     #2,d2           ;place the a2 address into the int. vector
        addi.l  #$100,d2        ;interrupt vector addr = (4 * int) + $000100
        move.l  d2,a1           ;transfer the calculated address to a register
        move.l  a2,(a1)         ;...that can act upon it thus!<--vector init'ed
        bsr.b   enabint         ;enable interrupts
        movem.l (sp)+,d0-d2/a0-a2       ;restore affected registers
        rts

**************************************************************************
*                                                                        *
*               disable an mfp interrupt via GEMDOS                      *
*                                                                        *
*       entry                                                            *
*                                                                        *
*       void    jdisint(numint)                                          *
*       word    numint                                                   *
*                                                                        *
**************************************************************************

        .globl  jdisint

jdisint move.w  4(sp),d0
        andi.l  #$f,d0          ;to ensure masking of 0-$f

**************************************************************************
*               interrupt disable routine                                *
**************************************************************************

disint
        movem.l d0-d1/a0-a1,-(sp)       ;save affected registers
        lea     mfp,a0          ;set mfp chip address pointer
        lea     imra(a0),a1     ;set a1 for the mskoff routine
        bsr.b   bselect         ;generate the appropriate bit to clear
        bclr    d1,(a1)         ;and clear the bit...
        lea     iera(a0),a1     ;set a1 for another mskoff call
        bsr.b   bselect
        bclr    d1,(a1)         ;and clear the bit...
        lea     ipra(a0),a1     ;yet again...
        bsr.b   bselect
        bclr    d1,(a1)         ;and clear the bit...
        lea     isra(a0),a1     ;now set up to check for interrupts in progress
        bsr.b   bselect         ;get proper a/b version...
        bclr    d1,(a1)
        movem.l (sp)+,d0-d1/a0-a1       ;restore affected registers
        rts

**************************************************************************
*                                                                        *
*       enable/re-enable an mfp interrupt via GEMDOS                     *
*                                                                        *
*       entry                                                            *
*                                                                        *
*       void    jenabint(numint)                                         *
*       word    numint                                                   *
*                                                                        *
**************************************************************************

        .globl  jenabint

jenabint
        move.w  4(sp),d0
        andi.l  #$f,d0          ;to ensure masking of 0-$f
```

```
**************************************************************************
*               enable interrupt routine                                 *
**************************************************************************
enabint
        movem.l d0-d1/a0-a1,-(sp)       ;save affected registers
        lea     mfp,a0          ;set mfp chip address pointer
        lea     iera(a0),a1     ;set up to enable the interrupt enable bit
        bsr.b   bselect
        bset    d1,(a1)         ;and set the bit...
        lea     imra(a0),a1     ;set up to enable the interrupt enable bit
        bsr.b   bselect
        bset    d1,(a1)         ;and set the bit...
        movem.l (sp)+,d0-d1/a0-a1       ;restore affected registers
        rts

**************************************************************************
*                                                                        *
*       the following routine generates the appropriate bset/bclr #      *
*       for the interrupt # specified in d0.    valid interrupt #'s are *
*       0 --> 15 as shown in the 68901 chip specification.  It also      *
*       selects between the ixra and the ixrb version of the register    *
*       as is appropriate.                                               *
*                                                                        *
*       entry   d0 - contains the interrupt number                       *
*               a1 - contains the pointer to the "ixra" version of       *
*                    the interrupt byte to mask                          *
*       exit            d0 - same as upon entry                          *
*                       d1 - contains the number of the bit              *
**************************************************************************

bselect
        move.b  d0,d1                   ;copy d0 to d1 for scratch work
        cmpi.b  #$8,d0                  ;see if desired int # >= 8...
        blt.b   skip0                   ;...and branch if it ain't...
        subq    #$8,d1                  ;adjust for using ixrb instead
skip0   cmpi.b  #$8,d0                  ;see if desired int # >= 8...
        bge.b   skip1                   ;...and branch if it is...
        addq    #$2,a1                  ;adjust for using ixrb instead
skip1   rts

        .page
        .text
rs232ptr
        lea     rbufrec,a0              ;point to current output buffer record
        lea     mfp,a1
        rts


rs232ibuf
        move.w  ibuftail(a0),d2
        move.w  ibufhead(a0),d3
        cmp.w   d3,d2                   ;is head-pointer > tail-pointer
        bhi.b   rb1                     ;no...
        add.w   ibufsiz(a0),d2  ;yes...buffer used=bufsiz+tail-head
rb1     sub.w   d3,d2                   ;obtain tail-head value
```

```
        rts

rtschk  btst.b  #$1,rsmode(a0)  ;check if we're using control lines
        beq.b   rtsexit         ;no...no need to assert rts on
        bsr     rtson           ;yes...turn on rts signal
rtsexit rts

*************************************************************************
*               putchar routine for rs-232 port                         *
*                                                                       *
*       this routine transfers characters to a output queue that is     *
*       emptied by an automatic interrupt routine.  the interrupt       *
*       routine handles the actual transfer of the character to the i/o *
*       port.                                                           *
*                                                                       *
*       entry                                                           *
*               d1 - contains character to transfer                     *
*       exit                                                            *
*               d0 - contains "0" for successful transfer, "xoff"       *
*                       for full buffer and no transfer                 *
*               carry bit clear - good transfer                         *
*               carry bit set - error condition                         *
*                                                                       *
*************************************************************************

rs232put
        move    sr,-(sp)        ;save sr
        ori     #$700,sr
        bsr.b   rs232ptr        ;point to current output buffer record

        btst.b  #$0,rsmode(a0)  ;are we using xon/xoff flow control?
        beq.b   rp0             ;no...

        tst.b   txoff(a0)       ;if non-zero then xon is in effect!
        bne.b   rp1             ;whether we're full or not, it's all the same!!

rp0     btst.b  #$7,tsr(a1)
        beq.b   rp1             ;buffer is full so keep char in circular buffr

        move.w  obufhead(a0),d2
        cmp.w   obuftail(a0),d2 ;head=tail?
        bne.b   rp1             ;yes...

        move.b  d1,udr(a1)      ;write a byte to transmit
        bra.b   rp3

rp1     move.w  obuftail(a0),d2 ;get current tail pointer offset from buffer
        bsr     wrapout         ;check for wrap of pointer
        cmp.w   obufhead(a0),d2 ;head=tail?
        beq.b   rp2             ;yes...no buffer space left
        move.l  obufptr(a0),a1  ;get current available buffer storage location
        move.b  d1,0(a1,d2)     ;store char to the buffer
        move.w  d2,obuftail(a0) ;store new tail pointer to buffer record

rp3     bsr     rtschk          ;do we turn on RTS signal line?
        move    (sp)+,sr
        andi.b  #$fe,ccr        ;indicate carry clear/good transfer
        rts                     ;done for now

rp2     bsr     rtschk          ;do we turn on RTS signal line?
        move    (sp)+,sr
        ori.b   #$1,ccr
        rts                     ;done for now

**************************************************************************
*               getchar routine for rs-232 port                          *
*                                                                        *
*       this routine transfers characters from a input queue that is     *
*       filled by an automatic interrupt routine.  the interrupt         *
*       routine handles the actual transfer of the character from the    *
*       i/o port.                                                        *
*                                                                        *
*       entry                                                            *
*               a0 - contains pointer to device buffer record            *
*       exit                                                             *
*               d0 - contains character if carry bit clear               *
*               if carry bit set then error condition                    *
*                                                                        *
**************************************************************************

rs232get
        move    sr,-(a7)        ;protect this upcoming test
        ori     #$700,sr

        bsr     rs232ptr        ;point to current output buffer record

        move.w  ibufhead(a0),d1 ;get current head pointer offset from buffer
        cmp.w   ibuftail(a0),d1 ;head=tail?
        beq.b   rg5             ;yes
        bsr     wrapin          ;check for wrap of pointer
        move.l  ibufptr(a0),a1  ;get base address of buffer
        moveq   #$0,d0          ;clear out  'd0'!
        move.b  0(a1,d1),d0     ;get character
        move.w  d1,ibufhead(a0) ;store new head pointer to buffer record

        move    (a7)+,sr
        andi.b  #%11111110,ccr  ;clear carry flag for normal return
        bra.b   rg4

rg5     move    (a7)+,sr
        ori.b   #$01,ccr        ;set carry for error condition just in case...

*       check rxoff flag and if set, see if low water mark is reached
*       if low watermark is reached, turn off rxoff flag and send a ctrl-q

rg4
        btst.b  #$0,rsmode(a0)  ;are we using xon/xoff flow control?
        beq.b   rg1             ;no...

        tst.b   rxoff(a0)       ;check for a current receiver xon situation
        beq.b   rg1             ;xon so continue...
```

```
* now check for lowwater mark triggering of flow-control

        bsr     rs232ibuf         ;get amount of input buffer used
        cmp.w   ibuflow(a0),d2    ;is amount consumed = lowmark?
        bne.b   rg1               ;no...

        move.b  #ctrlq,d1         ;setup rs232put/txrint to send a ctrl-q
        bsr     rs232put
        clr.b   rxoff(a0)         ;turn off rxoff flag byte

rg1     rts

*************************************************************************
*                                                                       *
*               receiver buffer full interrupt routine                  *
*                                                                       *
*               grabs data from the rs-232 receiver port                *
*                                                                       *
*************************************************************************

rcvrint
        movem.l d0-d3/a0-a2,-(sp)       ;save affected registers
        bsr     rs232ptr          ;point to current output buffer record

        move.b  rsr(a1),rsrbyte(a0)     ;do the required rsr read before
*                                       ;the udr read!
        btst.b  #7,rsrbyte(a0)    ;do rcvr buffer full flag test
        beq     ri8               ;branch should never be taken! means that the
*                                 ;wrong interrupt was called...should have been
*                                 ;the rcvr error interrupt procedure!
        btst.b  #$1,rsmode(a0)    ;check for currently using rts/cts/dtr
        beq.b   ri1               ;no...not currently in use
        bsr     rtsoff            ;yes...so clear rts to indicate we're busy
ri1     move.b  udr(a1),d0        ;get incoming data byte

*       now we do xon/xoff protocol check in case the byte we just got is
*       a ^s/^q.  we also check to see which mode we're in so that if we're in
*       binary or bypass mode (where the calling program handles the
*       handshaking!) we let the character into the buffer.  if we get either
*       character and are in xon/xoff protocol mode, we do not pass the
*       character along.  instead, we do the following

*       if we get a "^s" xoff, then we set the txoff flag byte to 1 to signal
*       to the txrint routine to stop transmitting.  the putchar routine to
*       the transmit buffer also checks the txoff byte and returns the carry
*       set if the byte may not be sent into the buffer.  see that routine for
*       a better explanation of how it handles txoff=1.

*       if we get a "^q" xon, then we reset the txoff flag byte to 1 to signal
*       to the txrint and the putchar routines to resume normal operation.

        btst.b  #$1,rsmode(a0)    ;check for currently using rts/cts/dtr
        bne.b   ri3               ;yes, so bypass xon/xoff flow control code...

        btst.b  #$0,rsmode(a0)    ;is the rs232 mode xon/xoff?
        beq.b   ri3               ;no...so process normally
        cmpi.b  #xon,d0           ;is the data an "xon" signal?
        bne.b   ri2               ;no...now check for xoff
        move.b  #$00,txoff(a0)    ;set to normal transmission status
        bra.b   ri8               ;abnormal exit condition!!

ri2     cmpi.b  #xoff,d0          ;check for xoff (^s) condition from host
        bne.b   ri3               ;neither xon/xoff value, must be normal data...
        move.b  #$ff,txoff(a0)    ;set to halted transmission to host
        bra.b   ri8               ;abnormal exit condition!!

ri3     move.w  ibuftail(a0),d1 ;get current tail pointer offset
        bsr     wrapin            ;do wrap of input pointer if needed
        cmp.w   ibufhead(a0),d1 ;head=tail?
        beq.b   ri8               ;yes...exit...

        move.l  ibufptr(a0),a2  ;get buffer pointer
        move.b  d0,0(a2,d1)       ;store the data
        move.w  d1,ibuftail(a0) ;store the new buftail pointer


* now check for highwater mark triggering of flow-control

        bsr     rs232ibuf         ;obtain amount of input buffer used
        cmp.w   ibufhigh(a0),d2 ;is amount consumed = highmark?
        bne.b   ri6               ;no...

* yes...send xoff to outside world
* set rxoff flag for the getchar and rcvrint routines

        btst.b  #$1,rsmode(a0)    ;check for currently using rts/cts/dtr
        bne.b   ri8               ;yes...exit without re-enabling DTR signal

        btst.b  #$0,rsmode(a0)    ;are we using xon/xoff flow control?
        beq.b   ri6               ;no...

        tst.b   rxoff(a0)         ;has a ctrl-s been sent yet?
        bne.b   ri6               ;yes...so don't send another
        move.b  #$ff,rxoff(a0)    ;means a ctrl-s has been sent to halt input
        move.b  #ctrls,d1         ;halt input from host
        bsr     rs232put

ri6     btst.b  #$1,rsmode(a0)    ;check for currently using rts/cts/dtr
        beq.b   ri8               ;no...not currently in use
        bsr     rtson             ;we're ready now for more data...yum! yum!

ri8     bclr.b  #$4,isra(a1)
        movem.l (sp)+,d0-d3/a0-a2         ;restore affected registers
        rte

***********************************************************************
*                                                                     *
*       transmit buffer empty interrupt routine                       *
*                                                                     *
***********************************************************************
```

```
txrint
        movem.l d2/a0-a2,-(sp)  ;save affected registers
        bsr     rs232ptr        ;point to current output buffer record

        btst.b  #$1,rsmode(a0)  ;are we using CTS/RTS flow control?
        bne.b   ti6             ;yes...get out of this routine and use CTSINT

        btst.b  #$0,rsmode(a0)  ;are we using xon/xoff flow control?
        beq.b   ti0             ;no...
        tst.b   txoff(a0)       ;if non-zero then xon is in effect!
        bne.b   ti6             ;whether we're full or not, it's all the same!!

ti0     move.b  tsr(a1),tsrbyte(a0)
        move.w  obufhead(a0),d2
        cmp.w   obuftail(a0),d2 ;head=tail?
        beq.b   ti6             ;yes...abnormal exit...
        bsr     wrapout         ;do wrap of input pointer if needed
        move.l  obufptr(a0),a2  ;get current buffer pointer
        move.b  0(a2,d2),udr(a1)        ;write a byte to transmit
        move.w  d2,obufhead(a0) ;store new head pointer

ti6     bclr.b  #$2,isra(a1)    ;turn off interrupt
        movem.l (sp)+,d2/a0-a2  ;restore affected registers
        rte

************************************************************************
*                                                                      *
*               Clear-To-Send interrupt routine                        *
*                                                                      *
*                                                                      *
************************************************************************

ctsint
        movem.l d2/a0-a2,-(sp)  ;save affected registers
        bsr     rs232ptr        ;point to current output buffer record

        btst.b  #$1,rsmode(a0)  ;are we using CTS/RTS flow control?
        beq.b   ctsexit         ;no...

        move.b  tsr(a1),tsrbyte(a0)
cts0    btst.b  #$7,tsrbyte(a0) ;is the transmit buffer empty yet?
        beq.b   cts0            ;no...continue looping

        move.w  obufhead(a0),d2
        cmp.w   obuftail(a0),d2 ;head=tail?
        beq.b   ctsempty        ;yes...abnormal exit...empty output buffer
        bsr     wrapout         ;do wrap of input pointer if needed
        move.l  obufptr(a0),a2  ;get current buffer pointer
        move.b  0(a2,d2),udr(a1)        ;write a byte to transmit
        move.w  d2,obufhead(a0) ;store new head pointer

ctsexit bclr.b  #$2,isrb(a1)    ;turn off interrupt
        movem.l (sp)+,d2/a0-a2  ;restore affected registers
        rte

ctsempty
```

```
        bra.b   ctsexit         ;exit via "ctsexit"

*************************************************************************
*       routines to handle tx or rx errors                              *
*************************************************************************

rxerror
        movem.l d0/a0-a1,-(sp)  ;save all registers
        bsr     rs232ptr                ;point to current output buffer record

        move.b  rsr(a1),rsrbyte(a0)     ;receiver status register
        move.b  udr(a1),d0              ;dummy read of data register
        bclr    #$3,isra(a1)
        movem.l (sp)+,d0/a0-a1  ;restore all registers
        rte

txerror
        movem.l a0-a1,-(sp)     ;save all registers
        bsr     rs232ptr                ;point to current output buffer record

        move.b  tsr(a1),tsrbyte(a0)     ;transmitter status register
        bclr    #$1,isra(a1)
        movem.l (sp)+,a0-a1     ;restore all registers
        rte

*************************************************************************
*                                                                       *
*               get device buffer record                                *
*                                                                       *
*       entry:                                                          *
*                                                                       *
*       long    iorec(device)                                           *
*       word    device                                                  *
*                                                                       *
*       returns pointer to the device's buffer record table             *
*                                                                       *
*               device - buffer identification number                   *
*                       0 - rs232                                       *
*                       1 - ikbd                                        *
*                       2 - midi                                        *
*                       3 - parallel                                    *
*                                                                       *
*               device table structure:                                 *
*                                                                       *
*               input buffer address            long                    *
*               input buffer size               word                    *
*               input buffer head               word                    *
*               input buffer tail               word                    *
*               input buffer low-water mark     word                    *
*               input buffer high-water mark    word                    *
*                                                                       *
*               output buffer address           long                    *
*               output buffer size              word                    *
*               output buffer head              word                    *
*               output buffer tail              word                    *
*               output buffer low-water mark    word                    *
*                 output buffer high-water mark    word                          *
*                                                                                *
**********************************************************************************

        .globl  iorec

iorec
        moveq   #0,d1
        move.w  4(sp),d1
        move.w  sr,-(sp)                     ;save sr for now
        ori.w   #$700,sr                     ;no interrupts for now
        lea     devtab,a2
        asl.l   #2,d1           ;x4=index into devtab space
        move.l  0(a2,d1.l),d0   ;get device bufrec pointer
        move.w  (sp)+,sr        ;save sr for now
        rts

devtab
        dc.l    rbufrec
        dc.l    kbufrec
        dc.l    mbufrec
*       dc.l    pbufrec         ;future consideration?

*********************************************************************************
*                                                                               *
*                 configure rs-232 port of MFP                                  *
*                                                                               *
*       entry:                                                                  *
*                                                                               *
*       void    rsconf(baudrate,flow,ucr,rsr,tsr,scr)                           *
*                                                                               *
*       word    baudrate - baud rate setting (value for timer D control        *
*                          and data registers)                                  *
*                          xxxxxxxx/xxxxxCCC/xxxxxxxx/DDDDDDDD                  *
*       word    flow -  flow control:    xxxxxxhs                               *
*                       h - cts/rts/dtr                                         *
*                       s - software xon/xoff                                   *
*                       1 - on, 0 - off                                         *
*       word    ucr -   MFP ucr register setting                                *
*       word    rsr -   MFP rsr register setting                                *
*       word    tsr -   MFP tsr register setting                                *
*       word    scr -   MFP scr register setting                                *
*                                                                               *
*********************************************************************************

        .globl  rsconf

rsconf
*       move.w  sr,-(sp)        ;save sr for now
        ori.w   #$700,sr        ;no interrupts for now

        bsr     rs232ptr
*
*       first, we grab the old ucr,rsr,tsr,scr contents
*
```

```
        movep.l ucr(a1),d7

*
*       next, we disable the receiver and transmitter enable bits
*
        moveq   #$0,d0          ;pre-init to zero
        move.b  d0,rsr(a1)      ;disable the receiver
        move.b  d0,tsr(a1)      ;disable the transmitter

*
*       set flow control mode(s)
*

        tst.w   $6(sp)          ;if -1 then don't change
        bmi.b   auxc1
        move.b  $7(sp),rsmode(a0)

*       set timer baud rate

        moveq   #0,d0
        moveq   #0,d2
auxc1   tst.w   $4(sp)          ;if -1 then don't change
        bmi.b   auxc2
        move.w  $4(sp),d1
        lea     baudctrl,a2     ;point to baudrate control register settings
        move.b  0(a2,d1.w),d0   ;get control mask
        lea     bauddata,a2     ;point to baudrate data register settings
        move.b  0(a2,d1.w),d2   ;get data reg value
        move.l  d0,d1           ;re-assign for "setimer" routine protocol
        moveq   #dtimer,d0              ;point to timer D
        bsr     setimer         ;set timer D to new baud rate

*       set rs-232 registers

auxc2   tst.w   $8(sp)          ;if -1 then don't change
        bmi.b   auxc3
        move.b  $9(sp),ucr(a1)
auxc3   tst.w   $a(sp)          ;if -1 then don't change
        bmi.b   auxc4
        move.b  $b(sp),rsr(a1)
auxc4   tst.w   $c(sp)          ;if -1 then don't change
        bmi.b   auxc5
        move.b  $d(sp),tsr(a1)
auxc5   tst.w   $e(sp)          ;if -1 then don't change
        bmi.b   auxc6
        move.b  $f(sp),scr(a1)
auxc6
*
*       finally we re-enable the receiver and transmitter enable bits
*
        moveq   #$1,d0          ;pre-init to one
        move.b  d0,rsr(a1)      ;enable the receiver
        move.b  d0,tsr(a1)      ;enable the transmitter

        move.l  d7,d0           ;move old contents of rs-232 registers to d0.l
```

```
*       move.w  (sp)+,sr        ;restore sr for now
        rts

*       baudrate table - control register setting

baudctrl
        dc.b    c19200,c9600,c4800,c3600
        dc.b    c2400,c2000,c1800,c1200
        dc.b    c600,c300,c200,c150
        dc.b    c134,c110,c75,c50

*       baudrate table - data register setting

bauddata
        dc.b    d19200,d9600,d4800,d3600
        dc.b    d2400,d2000,d1800,d1200
        dc.b    d600,d300,d200,d150
        dc.b    d134,d110,d75,d50

        .page
        .text

wrapin
        addq.w  #1,d1           ;i=h+1
        cmp.w   ibufsiz(a0),d1  ;? i>= current bufsiz?
        bcs.b   wi1             ;no...
        moveq   #$0,d1          ;wrap pointer
wi1     rts


wrapout
        addq.w  #1,d2           ;i=t+1
        cmp.w   obufsiz(a0),d2  ;? i>= current bufsiz?
        bcs.b   wo1             ;no...
        moveq   #$0,d2          ;wrap pointer
wo1     rts

        .page
        .text
***********************************************************************
*       this code handles the midi/keyboard interrupt exception      *
***********************************************************************

        .globl  midikey

midikey
        movem.l d0-d7/a0-a6,-(sp)       ;save all registers
        lea     $0,a5           ;address pointer to variable base
keymidi lea     mbufrec(a5),a0  ;point to midi buffer record
        lea     midi,a1         ;point to midi register base
        movea.l vmiderr(a5),a2  ;load in the jump vector
        bsr.b   astatus         ;goto general acia status check routine
        lea     kbufrec(a5),a0  ;point to ikbd buffer record
        lea     keyboard,a1     ;point to keyboard register base
        movea.l vkbderr(a5),a2  ;load in the jump vector
        bsr.b   astatus         ;goto general acia status check routine
        btst.b  #$4,gpip+mfp    ;check for pending interrupt occurance
        beq.b   keymidi         ;repeat this interrupt processing
        bclr.b  #$6,isrb+mfp    ;clear in-service bit
        movem.l (sp)+,d0-d7/a0-a6       ;restore all registers
        rte                     ;go back to what was happening!

astatus
        move.b  comstat(a1),d2  ;grab device status
        btst.l  #7,d2           ;make sure it was an interrupt request
        beq.b   aciaexit        ;nope...it's empty
        btst.l  #0,d2           ;see if receiver buffer is full
        beq.b   mk1             ;nope...it's empty
        movem.l d2/a0-a2,-(sp)
        bsr.b   arcvrint        ;yes...get byte
        movem.l (sp)+,d2/a0-a2
mk1     andi.b  #%00100000,d2   ;mask off bits already tested
        beq.b   aciaexit        ;see if any other status bits are on...
        move.b  iodata(a1),d0   ;grab data byte from acia data register
        jmp     (a2)            ;yes so branch to pre-inited error subroutine
aciaexit
        rts

************************************************************************
*                                                                      *
*               acia receiver buffer full interrupt routine            *
*                                                                      *
************************************************************************

        .globl  arcvrint

arcvrint
        move.b  iodata(a1),d0   ;grab data byte from acia data register
        cmpa.l  #kbufrec,a0
        bne     midibyte        ;don't treat midi acia data as anything other
*                               ;than as pure data...
        tst.b   kstate(a5)
        bne.b   ML3

        cmpi.b  #$f6,d0
        bcs     itsakey         ;branch early if it is not a ikbd header!
        subi.b  #$f6,d0         ;generate true index into tables now
        andi.l  #$ff,d0         ;clear high 3 bytes for indexing
        lea     ikbdev,a3       ;point to ikbd device state codes
        move.b  0(a3,d0),kstate(a5)     ;set ikbd state
        lea     ikbdlen,a3      ;point to ikbd device buffer length table
        move.b  0(a3,d0),kindex(a5)     ;set ikbd device index counter
        addi.w  #$f6,d0         ;re-constitute original value
        cmpi.b  #$f8,d0
        blt.b   ML8
        cmpi.b  #$fb,d0
        bgt.b   ML8
        move.b  d0,mousebuf(a5)
ML8     rts

ikbdev  dc.b    statks,amouse,rmouse,rmouse,rmouse,rmouse
        dc.b    clock,joyall,joy0,joy1
```

```
ikbdlen dc.b    statdex,amdex,rmdex-1,rmdex-1,rmdex-1,rmdex-1
        dc.b    clkdex,joyadex,joydex,joydex

ML3
        cmpi.b  #joy0,kstate(a5)
        bcc     ML35                ;a joystick 0/1 record byte, not both!
        lea     ikbdparams,a2       ;point to ikbd subsystem parameters table
        moveq   #$0,d2
        move.b  kstate(a5),d2       ;load to generate longword offset
        subq.b  #$1,d2              ;kstate(a5)=1 to 5/ table index is 0 to 4
        asl     d2                  ; x2
        add.b   kstate(a5),d2       ; +1
        subq.b  #$1,d2              ;kstate(a5)=1 to 5/ table index is 0 to 4
        asl     #2,d2               ; x4

        movea.l 0(a2,d2),a0         ;load in subsystem's record pointer
        movea.l 4(a2,d2),a1         ;load in subsystem's index base+record pointer
        movea.l 8(a2,d2),a2         ;load in subsystem's pointer variable that
*                                   ;contains the pointer to the subsystem's
*                                   ;interrupt routine...
        movea.l (a2),a2
        moveq   #$0,d2              ;clear out 'd2' for address manipulation
        move.b  kindex(a5),d2
        suba.l  d2,a1
        move.b  d0,(a1)
        sub.b   #1,kindex(a5)
        tst.b   kindex(a5)
        bne.b   ML1
ikserve move.l  a0,-(sp)            ;stuff buffer pointer to stack
        jsr     (a2)                ;go service the subsystem interrupt routine
        addq    #$4,sp              ;re-adjust stack
        clr.b   kstate(a5)                  ;reset ikbd state
ML1     rts

ikbdparams
        dc.l    statrec
        dc.l    statdex+statrec
        dc.l    statintvec

        dc.l    amrec
        dc.l    amdex+amrec
        dc.l    msintvec

        dc.l    mousebuf
        dc.l    rmdex+mousebuf
        dc.l    msintvec

        dc.l    clkrec
        dc.l    clkdex+clkrec
        dc.l    clkintvec

        dc.l    joyrec
        dc.l    joyadex+joyrec
        dc.l    joyintvec

ML35
```

```
        move.l  #joyrec+1,d1
        add.b   kstate(a5),d1   ;kstate(a5) reflects joy0 or joy1 state
        subi.b  #joy0,d1
        move.l  d1,a2           ;create index to joyrec table for record byte
        move.b  d0,(a2)
        movea.l joyintvec(a5),a2        ;get user's joystick interrupt routine adr
        lea     joyrec(a5),a0   ;send along address of joystick data
        bra.b   ikserve

itsakey
        move.b  kbshift(a5),d1  ;load in kbshift(a5) for manipulation...
* check the special keys
        cmpi.b  #$2A,d0         ;left shift?
        bne.b   ari2
        bset    #KBLSH,d1
        bra.b   ari10
ari2    cmpi.b  #$AA,d0
        bne.b   ari3
        bclr    #KBLSH,d1
        bra.b   ari10
ari3    cmpi.b  #$36,d0         ;right shift
        bne.b   ari4
        bset    #KBRSH,d1
        bra.b   ari10
ari4    cmpi.b  #$B6,d0
        bne.b   ari5
        bclr    #KBRSH,d1
        bra.b   ari10
ari5    cmpi.b  #$1D,d0         ;CTRL
        bne.b   ari6
        bset    #KBCTL,d1
        bra.b   ari10
ari6    cmpi.b  #$9D,d0
        bne.b   ari7
        bclr    #KBCTL,d1
        bra.b   ari10
ari7    cmpi.b  #$38,d0         ;ALT
        bne.b   ari8
        bset    #KBALT,d1
        bra.b   ari10
ari8    cmpi.b  #$B8,d0
        bne.b   ari9
        bclr    #KBALT,d1
        bra.b   ari10
ari9    cmpi.b  #$3A,d0         ;CAPS LOCK
        bne.b   ari11
        btst.b  #0,conterm(a5)
        beq.b   ari9a           ;no click please!
        move.l  #keyclk,cursnd(a5)
        move.b  #0,timer(a5)
ari9a   bchg    #KBCL,d1        ;toggle CAPS LOCK state
ari10   move.b  d1,kbshift(a5)  ;restore new kbshift(a5) value
        rts                     ;ignore CAPS LOCK break
ari11   btst.l  #7,d0           ;is it a break code?
        bne.b   ari12           ;no... a break code!
        tst.b   keyrep(a5)              ;yes
        bne.b   ari15
        move.b  d0,keyrep(a5)   ;save for repeat purpose
        move.b  cdelay1,kdelay1(a5)
        move.b  cdelay2,kdelay2(a5)
        bra.b   ari16
ari15   move.b  #0,kdelay1(a5)
        move.b  #0,kdelay2(a5)
        bra.b   ari16
ari12   tst.b   keyrep(a5)
        beq.b   ari18
        moveq   #0,d1
        move.b  d1,keyrep(a5)
        move.b  d1,kdelay1(a5)
        move.b  d1,kdelay2(a5)

ari18   cmpi.b  #$c7,d0         ;is it a "home" break-code?
        beq.b   ari18a          ;yes...allow it to pass
        cmpi.b  #$d2,d0         ;is it a "insert" break-code?
        bne     ari14           ;no...regular break junk...
ari18a  btst.b  #KBALT,kbshift(a5)      ;early "ALT" test to prevent double "nulls
        beq     ari14           ;no ALT...so exit now...

ari16   btst.b  #0,conterm(a5)
        beq.b   ari16a          ;no click please!
        move.l  #keyclk,cursnd(a5)
        move.b  #0,timer(a5)

ari16a  move.l  a0,-(sp)        ;store kbufrec pointer

        moveq   #$0,d1
        move.b  d0,d1

        movea.l skeytran(a5),a0
        andi.w  #$7F,d0
        btst.b  #KBCL,kbshift(a5)
        beq.b   conin21
        movea.l skeycl(a5),a0
conin21 btst.b  #KBRSH,kbshift(a5)
        bne.b   conin22
        btst.b  #KBLSH,kbshift(a5)
        beq.b   conin23
conin22 cmpi.b  #$3b,d0         ;see if a possible function key
        bcs.b   conin22a        ;unsigned less than lowest function scancode
        cmpi.b  #$44,d0         ;see if a possible function key
        bhi.b   conin22a        ;unsigned greater than highest function scan
        addi.w  #$19,d1         ;add to change to GSX standard
        moveq   #$0,d0          ;change to GSX standard
        bra     conin25
conin22a
        movea.l skeyshif(a5),a0
conin23 move.b  (a0,d0.w),d0
        btst.b  #KBCTL,kbshift(a5)      ;is the control key down?
        beq.b   conin24a
        cmpi.b  #cr,d0          ;is it a carriage return?
        bne.b   conin23a
        moveq   #lf,d0          ;change to a linefeed according to GSX spec...
        beq.b   conin24
conin23a
        cmpi.b  #$47,d1         ;convert CONTROL-home to gsx standard
        bne.b   conin23b        ;by adding #$30...
        addi.w  #$30,d1
        bra     conin25
conin23b
        cmpi.b  #$4b,d1         ;convert CONTROL-left arrow to gsx standard
        bne.b   conin23c
        moveq   #$73,d1         ;change according to gsx spec
        moveq   #$0,d0
        bra     conin25
conin23c
        cmpi.b  #$4d,d1         ;convert CONTROL-right arrow to gsx standard
        bne.b   conin24
        moveq   #$74,d1         ;change according to gsx spec
        moveq   #$0,d0
        bra     conin25
conin24
        andi.w  #$01F,d0        ;yep, so CTRLize the key
        bra     conin25
conin24a
        btst.b  #KBALT,kbshift(a5)      ;is the alt key down?
        beq     conin25

        ifeq    COUNTRY-GERMANY

        cmpi.b  #$1a,d1         ;is it a ALT-umlaut?
        bne.b   altger1         ;no...
        move.b  #$40,d0         ;put in '@', then check the shift keys
        move.b  kbshift(a5),d2  ;grab current setting
        andi.b  #$3,d2          ;KBRSH+KBLSH bits
        beq     conin25         ;process it as unshifted
        move.b  #$5c,d0         ;put in '\', instead...it's a alt-shift umlaut!
        bra     conin25         ;process it
altger1 cmpi.b  #$27,d1         ;is it a ALT-
        bne.b   altger2         ;no...
        move.b  #$5b,d0         ;put in '[', then check the shift keys
        move.b  kbshift(a5),d2  ;grab current setting
        andi.b  #$3,d2          ;KBRSH+KBLSH bits
        beq     conin25         ;process it as unshifted
        move.b  #$7b,d0         ;put in '{', instead...it's a alt-shift umlaut!
        bra     conin25         ;process it
altger2 cmpi.b  #$28,d1         ;is it a ALT-
        bne.b   outside         ;no...
        move.b  #$5d,d0         ;put in ']', then check the shift keys
        move.b  kbshift(a5),d2  ;grab current setting
        andi.b  #$3,d2          ;KBRSH+KBLSH bits
        beq     conin25         ;process it as unshifted
        move.b  #$7d,d0         ;put in '}', instead...it's a alt-shift umlaut!
        bra     conin25         ;process it

        endc

        ifeq    COUNTRY-FRANCE
        cmpi.b  #$1a,d1         ;is it a ALT-^?
        bne.b   altfr1          ;no...
        move.b  #$5b,d0         ;put in '[', then check the shift keys
        move.b  kbshift(a5),d2  ;grab current setting
        andi.b  #$3,d2          ;KBRSH+KBLSH bits
        beq     conin25         ;process it as unshifted
        move.b  #$7b,d0         ;put in '{', instead...it's a alt-shift ^
        bra     conin25         ;process it
altfr1  cmpi.b  #$1b,d1         ;is it a ALT-$?
        bne.b   altfr2          ;no...
        move.b  #$5d,d0         ;put in ']', then check the shift keys
        move.b  kbshift(a5),d2  ;grab current setting
        andi.b  #$3,d2          ;KBRSH+KBLSH bits
        beq     conin25         ;process it as unshifted
        move.b  #$7d,d0         ;put in '}', instead...it's a alt-shift $
        bra     conin25         ;process it
altfr2  cmpi.b  #$28,d1         ;is it a ALT-
        bne.b   altfr3          ;no...
        move.b  #$5c,d0         ;put in '\', then check the shift keys
        move.b  kbshift(a5),d2  ;grab current setting
        andi.b  #$3,d2          ;KBRSH+KBLSH bits
        beq     conin25         ;process it as unshifted
        move.b  #$00,d0         ;put in 'NUL',instead...it's a alt-shift
        bra     conin25         ;process it
altfr3  cmpi.b  #$2b,d1         ;is it a ALT-#?
        bne.b   outside         ;no...
        move.b  #$40,d0         ;put in '@', then check the shift keys
        move.b  kbshift(a5),d2  ;grab current setting
        andi.b  #$3,d2          ;KBRSH+KBLSH bits
        beq     conin25         ;process it as unshifted
        move.b  #$7e,d0         ;put in '|', instead...it's a alt-shift #
        bra     conin25         ;process it

        endc


outside cmpi.b  #$62,d1         ;is it an "alt help" signal to dump the screen?
        bne.b   alt15a          ;no...
        addq.w  #1,_dumpflg(a5) ;yes...switch the signal flag on!...
        movea.l (sp)+,a0        ;restore kbufrec pointer
        bra     ari14           ;...and exit
*
*       check the alt-insert/alt-home key make/break combinations, first
*
alt15a  lea     mauskey1,a2     ;get pointer to first alt. mouse scancode table
        moveq   #3,d2           ;create countdown
mkloop1 cmp.b   0(a2,d2),d1     ;is table's scancode value = current value?
        beq     keymaus1        ;yes...go preprocess it...
        dbra    d2,mkloop1      ;no...loop back to check next table value

        cmpi.b  #$48,d1         ;is it an up arrow?
        bne.b   alt11
        move.b  #$0,d1          ;x value for up arrow
        move.b  #$-8,d2         ;y value for up arrow
        move.b  kbshift(a5),d0  ;grab current setting
        andi.b  #$3,d0          ;KBRSH+KBLSH bits
        beq     keymaus
        move.b  #$-1,d2         ;y value for up arrow
        bra     keymaus
alt11   cmpi.b  #$4b,d1         ;is it an left arrow?
        bne.b   alt12
        move.b  #$0,d2          ;y value for left arrow
        move.b  #$-8,d1         ;x value for left arrow
        move.b  kbshift(a5),d0  ;grab current setting
        andi.b  #$3,d0          ;KBRSH+KBLSH bits
        beq     keymaus
        move.b  #$-1,d1         ;x value for left arrow
        bra     keymaus
alt12   cmpi.b  #$4d,d1         ;is it an right arrow?
        bne.b   alt13
        move.b  #$8,d1          ;x value for right arrow
        move.b  #$0,d2          ;y value for right arrow
        move.b  kbshift(a5),d0  ;grab current setting
        andi.b  #$3,d0          ;KBRSH+KBLSH bits
        beq     keymaus
        move.b  #$1,d1          ;x value for right arrow
        bra     keymaus
alt13   cmpi.b  #$50,d1         ;is it an down arrow?
        bne.b   alt14
        move.b  #$0,d1          ;x value for down arrow
        move.b  #$8,d2          ;y value for down arrow
        move.b  kbshift(a5),d0  ;grab current setting
        andi.b  #$3,d0          ;KBRSH+KBLSH bits
        beq     keymaus
        move.b  #$1,d2          ;y value for down arrow
        bra     keymaus
alt14   cmpi.b  #$2,d1
        bcs.b   alt1            ;not >= the '1' key scancode
        cmpi.b  #$d,d1
        bhi.b   alt1            ;not <= the '=' key scancode
        addi.b  #$76,d1         ;scancode is a key between '1' key and '=' key
        bra.b   alt2
alt1    cmpi.b  #$41,d0         ;is the key an ascii 'A' or greater?
        bcs.b   alt3            ;no...skip to check if 'a'-'z'...
        cmpi.b  #$5a,d0         ;is the key an ascii 'Z' or less?
        bhi.b   alt3            ;no...skip to check if 'a'-'z'...
alt2    moveq   #$0,d0
        bra.b   conin25
alt3    cmpi.b  #$61,d0         ;is the key an ascii 'a' or greater?
        bcs.b   conin25         ;no...skip to finish normal processing
        cmpi.b  #$7a,d0         ;is the key an ascii 'z' or less?
        bhi.b   conin25         ;no...skip to finish normal processing
        bra.b   alt2
conin25 and.w   #$7f,d1
        asl.w   #$8,d1          ;shift the scan code to the word's high byte
        add.w   d1,d0           ;form the outgoing word

        movea.l (sp)+,a0        ;restore kbufrec pointer

        move.w  ibuftail(a0),d1 ;get current tail pointer offset
        addq    #2,d1           ;index = tail + 2
        cmp.w   ibufsiz(a0),d1  ;check to see if buffer should wrap
        bcs.b   ari13               ;no...
        moveq   #$0,d1              ;wrap pointer
ari13   cmp.w   ibufhead(a0),d1 ;head=tail?
        beq.b   ari14               ;yes
        move.l  ibufptr(a0),a2  ;get buffer pointer
        move.w  d0,0(a2,d1)         ;store the data
        move.w  d1,ibuftail(a0) ;store the new buftail pointer
ari14   rts

midibyte
        movea.l midivec(a5),a2  ;get contents of midivec for indirect branch
        jmp     (a2)                ;jump to midi interrupt handler

sysmidi move.w  ibuftail(a0),d1 ;get current tail pointer offset
        addq    #1,d1               ;index = tail + 1
        cmp.w   ibufsiz(a0),d1  ;check to see if buffer should wrap
        bcs.b   mi13                ;no...
        moveq   #$0,d1              ;wrap pointer
mi13    cmp.w   ibufhead(a0),d1 ;head=tail?
        beq.b   mi14                ;yes
        move.l  ibufptr(a0),a2  ;get buffer pointer
        move.b  d0,0(a2,d1)         ;store the data
        move.w  d1,ibuftail(a0) ;store the new buftail pointer
mi14    rts

keymaus1
        moveq   #KBMRB,d3           ;pre-init to "keyboard" right mouse button
        btst    #4,d1               ;see if it is a left or right button...
        beq.b   kym1                ;it's a right button ($47/$c7)
        moveq   #KBMLB,d3           ;it's a left button ($52/$d2)
kym1    btst    #7,d1               ;see if it is a make or break action
        beq.b   kym2                ;it's a set button action (make code)
        bclr    d3,kbshift(a5)  ;it's a clear button action (break code)
        bra.b   kym3                ;go to further pre-init action...
kym2    bset    d3,kbshift(a5)  ;it's a set button action (set code)
kym3    moveq   #$0,d1
        moveq   #$0,d2
*
*       finish up at the actual pseudo mouse routine
*

keymaus
        lea     kmbuf(a5),a0        ;point to key-emulating mouse buffer
        movea.l msintvec(a5),a2 ;grab mouse interrupt vector
        clr.l   d0
        move.b  kbshift(a5),d0  ;get current button status
        lsr.b   #KBMRB,d0           ;shift right button bit to 'd0'
        addi.b  #$f8,d0             ;add relative mouse header
        move.b  d0,0(a0)            ;store in first byte of record buffer
        move.b  d1,1(a0)            ;store x value in second byte of record buffer
        move.b  d2,2(a0)            ;store y value in third byte of record buffer
        jsr     (a2)
        movea.l (sp)+,a0            ;restore kbufrec pointer
        rts

mauskey1
```

```
        dc.b    $47
        dc.b    $c7
        dc.b    $52
        dc.b    $d2

        .page
        .text

************************************************************************
*                                                                      *
*       protocol for accessing a gi sound chip register               *
*                                                                      *
*       this bios call must be accessed in supervisor state           *
*       because it affects the 'sr' register                          *
*                                                                      *
*       entry                                                          *
*                                                                      *
*       void    giaccess(data,register)                                *
*       word    data,register                                          *
*                                                                      *
*               data -- data register read/write data                  *
*                                                                      *
*               register -- chip register to select                    *
*               d1 = #$0000     ;selects read operation of the register *
*               d1 = #$80 .or .xx       ;selects write xx to register  *
*               example write to portb - $80 .or. $0f = $8f            *
*                                                                      *
*       exit                                                           *
*       read operations                                                *
*       d0.b -- data register contains byte of date                    *
*       write operations                                               *
*       d0.b -- data register contains a verification of written data  *
*                                                                      *
*                                                                      *
************************************************************************

        .globl  giaccess

giaccess
        move.w  4(sp),d0
        move.w  6(sp),d1
gientry
        move    sr,-(a7)
        ori     #$0700,sr
        movem.l d1-d2/a0,-(a7)  ;save affected registers
        lea     giselect,a0     ;init desired gi register addr
        move.b  d1,d2           ;make a copy to test for read or write
        andi.b  #$f,d1          ;turn off any extraneous bits
        move.b  d1,(a0)         ;select register
        asl.b   #1,d2           ;shift once for carry bit detection
        bcc.b   giread          ;carry clear, so do a read operation
giwrit  move.b  d0,2(a0)        ;init the memory location
giread  moveq   #$0,d0          ;clear out register
        move.b  (a0),d0         ;grab the data from the gi register
        movem.l (a7)+,d1-d2/a0  ;restore affected registers
        move    (a7)+,sr
        rts                             ;return with data in d0

************************************************************************
*               routine to turn off the rts signal                     *
************************************************************************
        .globl  rtsoff

rtsoff
        moveq   #%00001000,d2
        bra.b   onbit

************************************************************************
*               routine to turn on the rts signal                      *
************************************************************************
        .globl  rtson

rtson
        moveq   #%11110111,d2
        bra.b   offbit

************************************************************************
*               routine to turn off the dtr signal                     *
************************************************************************
        .globl  dtroff
dtroff
        moveq   #%00010000,d2
        bra.b   onbit

************************************************************************
*               routine to turn on the dtr signal                      *
************************************************************************
        .globl  dtron
dtron
        moveq   #%11101111,d2
        bra.b   offbit

************************************************************************
*                                                                      *
*       routine to set any bit in the gi port a area                   *
*                                                                      *
*       entry                                                          *
*                                                                      *
*       void    ongibit(bitnum)                                        *
*       word    bitnum                                                 *
*                                                                      *
*               bitnum - byte size bit mask with desired bit set to "1" *
*                                                                      *
************************************************************************

        .globl  ongibit

ongibit
        moveq   #$0,d2
        move.w  4(sp),d2
onbit   movem.l d0-d2,-(a7)
```

```
        move    sr,-(a7)
        ori     #$0700,sr
        moveq   #porta,d1       ;get ready to read in the port a contents
        move.l  d2,-(a7)
        bsr.b   gientry         ;go get it...
        move.l  (a7)+,d2
        or.b    d2,d0           ;set bit(s) on
        moveq   #porta+$80,d1   ;setup to write to port a
        bsr.b   gientry         ;go set it and return
        move    (a7)+,sr
        movem.l (a7)+,d0-d2
        rts

************************************************************************
*                                                                      *
*       routine to clear any bit in the gi port a area                 *
*                                                                      *
*       entry                                                          *
*                                                                      *
*       void    offgibit(bitnum)                                       *
*       word    bitnum                                                 *
*                                                                      *
*               bitnum - byte size bit mask with desired bit set to "0" *
*                                                                      *
************************************************************************

        .globl  offgibit

offgibit
        moveq   #$0,d2
        move.w  4(sp),d2
offbit  movem.l d0-d2,-(a7)
        move    sr,-(a7)
        ori     #$0700,sr
        moveq   #porta,d1       ;get ready to read in the port a contents
        move.l  d2,-(a7)
        bsr.b   gientry         ;go get it...
        move.l  (a7)+,d2
        and.b   d2,d0           ;turn bit(s) off
        moveq   #porta+$80,d1   ;setup to write to port a
        bsr.b   gientry         ;go set it and return
        move    (a7)+,sr
        movem.l (a7)+,d0-d2
        rts

        .page
        .text

************************************************************************
*                                                                      *
*               EXTENDED RBP BIOS MOUSE INIT CALL                      *
*                                                                      *
*       entry:                                                         *
*                                                                      *
*       void    initmous(type,param,intvec)                            *
*       word    type                                                   *
*       long    param,intvec                                            *
*                                                                       *
*               type - key/abs/rel/off  mouse function requested        *
*                       4/  2/  1/  0   value                           *
*               param - address of parameter block                      *
*               intvec - mouse interrupt vector                         *
*                                                                       *
*                                                                       *
*       parameter block definition:                                     *
*                                                                       *
*       byte 0 - y=0 at top/bottom; if non-zero then y=0 at bottom      *
*               otherwise y=0 at top                                    *
*       byte 1 - parameter for set mouse buttons command                *
*       byte 2 - x threshold/scale/delta parameter                      *
*       byte 3 - y threshold/scale/delta parameter                      *
*                                                                       *
*       the following bytes are required for the absolute mouse only    *
*                                                                       *
*       byte 4 - xmsb for absolute mouse maximum position               *
*       byte 5 - xlsb for absolute mouse maximum position               *
*       byte 6 - ymsb for absolute mouse maximum position               *
*       byte 7 - ylsb for absolute mouse maximum position               *
*       byte 8 - xmsb for absolute mouse initial position               *
*       byte 9 - xlsb for absolute mouse initial position               *
*       byte a - ymsb for absolute mouse initial position               *
*       byte b - ylsb for absolute mouse initial position               *
*                                                                       *
*************************************************************************

        .globl  initmouse

initmouse
*       first we determine if the init is for a absolute, relative, or keycode
*       mouse action.

        tst.w   $4(sp)          ;turn mouse off?
        beq.b   im1             ;yes...disable mouse
        move.l  $a(sp),msintvec(a5)     ;init the mouse interrupt vector
        move.l  $6(sp),a3
        cmpi.w  #$1,$4(sp)      ;relative mouse request?
        beq.b   im2             ;yes...
        cmpi.w  #$2,$4(sp)      ;absolute mouse request?
        beq.b   im3             ;yes...
        cmpi.w  #$4,$4(sp)      ;keycode mouse request?
        beq.b   im4             ;yes...
        moveq   #$0,d0          ;error condition returned -- improper request
        rts
im1     moveq   #$12,d1         ;disable mouse
        bsr     ikbdput
        move.l  #xbtexit,msintvec(a5)   ;re-init the mouse interrupt vector
        bra.b   imexit
im2
        lea     transbuf(a5),a2 ;set transfer buffer pointer
        move.b  #$8,(a2)+       ;set to relative mouse
        move.b  #$b,(a2)+       ;set relative mouse threshold x,y
        bsr.b   setmouse
        moveq   #7-1,d3             ;set length of string -1 to transfer
        lea     transbuf(a5),a2     ;set transfer buffer pointer
        bsr     ikbdstr             ;do transfer to ikbd
        bra.b   imexit
im3
        lea     transbuf(a5),a2     ;set transfer buffer pointer
        move.b  #$9,(a2)+           ;set to absolute mouse
        move.b  4(a3),(a2)+         ;set xmsb max
        move.b  5(a3),(a2)+         ;set xlsb max
        move.b  6(a3),(a2)+         ;set ymsb max
        move.b  7(a3),(a2)+         ;set ylsb max
        move.b  #$c,(a2)+           ;set absolute mouse scale
        bsr.b   setmouse
        move.b  #$e,(a2)+           ;load initial absolute mouse position
        move.b  #$0,(a2)+           ;filler load
        move.b  8(a3),(a2)+         ;initial xmsb absolute mouse position
        move.b  9(a3),(a2)+         ;initial xlsb absolute mouse position
        move.b  $a(a3),(a2)+        ;initial ymsb absolute mouse position
        move.b  $b(a3),(a2)+        ;initial ylsb absolute mouse position
        moveq   #17-1,d3            ;set length of string -1 to transfer
        lea     transbuf(a5),a2     ;set transfer buffer pointer
        bsr     ikbdstr             ;do transfer to ikbd
        bra.b   imexit
im4
        lea     transbuf(a5),a2     ;set transfer buffer pointer
        move.b  #$a,(a2)+           ;set to mouse keycode mode
        bsr.b   setmouse
        moveq   #6-1,d3             ;set length of string -1 to transfer
        lea     transbuf(a5),a2     ;set transfer buffer pointer
        bsr     ikbdstr             ;do transfer to ikbd
imexit  moveq   #$-1,d0             ;set to true to indicate good init
        rts
setmouse
        move.b  2(a3),(a2)+         ;set x threshold/scale/delta
        move.b  3(a3),(a2)+         ;set y threshold/scale/delta
        moveq   #$10,d1             ;setup to determine if top/bottom
        sub.b   0(a3),d1            ;set y=0 at ?
        move.b  d1,(a2)+
        move.b  #$7,(a2)+           ;set mouse button action
        move.b  1(a3),(a2)+         ;mouse button parameter
        rts

************************************************************************
*                                                                      *
*               EXTENDED RBP BIOS TIMER INIT CALL                      *
*                                                                      *
*       entry:                                                         *
*                                                                      *
*       void    xbtimer(id,control,data,intvec)                        *
*       word    id,control,data                                        *
*       long    intvec                                                 *
*                                                                      *
*               intvec - timer interrupt vector                        *
*               control - timer's control setting                      *
*               data - timer's data register setting                   *
*               id - timer id   a-0, b-1, c-2, d-3                     *
*                                                                         *
*       Special Note:                                                     *
*                                                                         *
*       In the interest of preserving as many features for the user       *
*       in the future, timer A should be reserved for the end-user        *
*       or independent software vendor's application program.  System     *
*       software or those application needing just a "tick" should        *
*       constrain themselves to timer C, which is adequate for delay      *
*       and other timing uses.  Future hardware may or may not bring      *
*       out the timer A input line out...giving software developers       *
*       another useful aspect of the machine to utilize.                  *
*                                                                         *
*       The recommended usage of the timers is as follows:                *
*                                                                         *
*       Timer A - Reserved for end-users and stand-alone applications.    *
*       Timer B - Reserved for screen graphics, primarily.                *
*       Timer C - Reserved for system timing (GSX,GEM,DESKTOP,ET.AL).     *
*       Timer D - Reserved for baud rate control of RS-232 port,          *
*                 the interrupt vector is available to anyone.            *
*                                                                         *
***************************************************************************

        .globl  xbtimer

xbtimer
        moveq   #$0,d0
        moveq   #$0,d1
        moveq   #$0,d2
        move.w  $4(sp),d0
        move.w  $6(sp),d1
        move.w  $8(sp),d2
        bsr     setimer                 ;setup the timer
        tst.l   $a(sp)                  ;if >$7fffffff then skip and exit
        bmi.b   xbtexit
        movea.l $a(sp),a2               ;setup for initint call
        moveq   #$0,d1                  ;clear long
        lea     xbtim,a1                ;point to timer -> interrupt # translation tab
        andi.l  #$ff,d0                 ;mask off the highest three bytes in register
        move.b  0(a1,d0),d0             ;setup for initint call
        bsr     initint
xbtexit rts
xbtim   dc.b    $d,$8,$5,$4
        .even

***************************************************************************
*                                                                         *
*               KEYBOARD TRANSLATION TABLE CHANGE CALL                    *
*                                                                         *
*       entry:                                                            *
*                                                                         *
*       long    keytrans(unshift,shift,capslock)                          *
*       long    unshift,shift,capslock                                    *
*                                                                         *
*               -1 signifies no change to vector                          *
*                                                                         *
        exit:                                                             *
```

```
*               d0.l - returns pointer to beginning of                  *
*                       key translation address pointers                *
*               order of pointers is:                                   *
*               unshifted,shifted,caps-locked                           *
*               Note:  buffer space for each table should $80!!         *
*                                                                       *
*************************************************************************

        .globl  keytrans

keytrans
        tst.l   $4(sp)
        bmi.b   kt1
        move.l  $4(sp),skeytran(a5)
kt1     tst.l   $8(sp)
        bmi.b   kt2
        move.l  $8(sp),skeyshif(a5)
kt2     tst.l   $c(sp)
        bmi.b   kt3
        move.l  $c(sp),skeycl(a5)
kt3     move.l  #skeytran,d0
        rts

*************************************************************************
*                                                                       *
*               RESTORE BIOS KEYBOARD TRANSLATION TABLE                 *
*                                                                       *
*       entry:                                                          *
*                                                                       *
*       void    bioskeys()                                              *
*                                                                       *
*************************************************************************

        .globl  bioskeys

bioskeys
        move.l  #keytran,skeytran(a5)
        move.l  #keyshif,skeyshif(a5)
        move.l  #keycl,skeycl(a5)
        rts

*************************************************************************
*                                                                       *
*               RETURN IKBD SUBSYSTEM INTERRUPT TABLE POINTER           *
*                                                                       *
*       entry:                                                          *
*                                                                       *
*       void    dosound(ptr)                                            *
*       long    ptr     ;points to start of sound interpreter table     *
*                                                                       *
*                                                                       *
*************************************************************************

        .globl  dosound

dosound
```

```
        move.l  cursnd(a5),d0           ; return current status in D0.L
        move.l  4(sp),d1                ; if new ptr < 0, then just return
        bmi     ds_r                    ; (invalid ptr, so return)
        move.l  d1,cursnd(a5)           ; setup new sound ptr
        clr.b   timer(a5)               ; zap sound timer register
ds_r    rts

*************************************************************************
*                                                                       *
*               SET/RETURN PRINTER CONFIGURATION WORD                   *
*                                                                       *
*       entry:                                                          *
*                                                                       *
*       word    setprt(pconfig)                                         *
*       word    pconfig ;sets/gets printer information word             *
*                                                                       *
*                                                                       *
*************************************************************************

        .globl  setprt

setprt
        move.w  pconfig(a5),d0  ;get current config word before we change it
        tst.w   4(sp)           ;see if we don't change the word
        bmi.b   nosetp          ;don't set printer word
        move.w  4(sp),pconfig(a5)       ;set printer config word
nosetp  rts

*************************************************************************
*                                                                       *
*               SET/RETURN KEY REPEAT VALUES                            *
*                                                                       *
*       entry:                                                          *
*                                                                       *
*       word    kbrate(initial,repeat)                                  *
*       word    initial,repeat                                          *
*                                                                       *
*       initial determines the number of 50 hz cycles to wait before    *
*       a keyrepeat is to commence.  repeat determines the interval     *
*       between keyrepeats after the initial pause.                     *
*                                                                       *
*************************************************************************

        .globl  kbrate

kbrate
        move.w  cdelay1(a5),d0  ;get current initial/repeat values
        tst.w   4(sp)           ;see if we don't change the word
        bmi.b   kbrate1         ;don't set key repeat values
        move.w  4(sp),d1        ;set key repeat values
        move.b  d1,cdelay1(a5)  ;set initial delay
        tst.w   6(sp)           ;see if we don't change the word
        bmi.b   kbrate1         ;don't set key repeat values
        move.w  6(sp),d1        ;set key repeat values
        move.b  d1,cdelay2(a5)  ;set subsequent delay
kbrate1 rts
```

```
***********************************************************************
*                                                                     *
*               RETURN POINTER TO IKBD/MIDI INTERRUPT VECTORS         *
*                                                                     *
*       entry:                                                        *
*                                                                     *
*       long    ikbdvecs()                                            *
*               returns a pointer to the midi interrupt vector and    *
*               ikbd subsystem interrupt vector table.  the table     *
*               structure is as follows:                              *
*                                                                     *
*       midivec         ds.l    1       ;midi interrupt handler vector *
*       vkbderr         ds.l    1       ;keyboard error handler address *
*       vmiderr         ds.l    1       ;midi error handler address   *
*       statintvec      ds.l    1       ;ikbd status interrupt vector *
*       msintvec        ds.l    1       ;mouse interrupt vector       *
*       clkintvec       ds.l    1       ;realtime clk interrupt vector *
*       joyintvec       ds.l    1       ;joystick interrupt vector    *
*                                                                     *
*       note:   msintvec is modified via the initmouse system function *
*               call.  since gem uses this vector, modifying it can be *
*               fatal while running under gem.  clkintvec is used by  *
*               gemdos.  its pre-inited vector must be restored for   *
*               proper gemdos operation.  Caveat hacker!              *
*                                                                     *
*                                                                     *
***********************************************************************

        .globl  ikbdvecs

ikbdvecs
        move.l  #midivec,d0
        rts


***********************************************************************
*                                                                     *
*       C Timer interrupt routine to process the PSG sound table      *
*                                                                     *
***********************************************************************
*+ (lmd)
* timercint - timer c interrupt handler
* divide 200 Hz interrupt frequency to 50 hz, and do:
*       sound handler processing
*       key-repeat processing;
*       control-g bell and keyclick if enabled via sound handler
*       system timer-tick handoff.
*       updates:        tc_rot (every tick)
*
*       imports:        etv_timer (timer handoff vector)
*                       _timr_ms (timer calibration value)
*
*-

timercint
```

```
        add.l   #1,_hz_200          ;increment raw tick counter
        rol.w   tc_rot              ;rotate divisor bits
        bpl.b   t_punt              ;if not 4th interrupt, then return

        movem.l d0-d7/a0-a6,-(sp)

        lea     $0,a5               ;address pointer to variable base

        bsr.b   sndirq              ;process sounds...

        btst.b  #$1,conterm(a5) ;check for key repeat enabled
        beq.b   krexit              ;not enabled

*       process for repeat key function first because it can affect the sound
*       table if enabled and the user is 'using'...

        tst.b   keyrep(a5)
        beq.b   krexit
        tst.b   kdelay1(a5)
        beq.b   kr1
        subi.b  #1,kdelay1(a5)
        bne.b   krexit
kr1     subi.b  #1,kdelay2(a5)
        bne.b   krexit
        move.b  cdelay2(a5),kdelay2(a5)
        move.b  keyrep(a5),d0
        lea     kbufrec(a5),a0
        bsr     ari16               ;repeat key stroke and stuff into buffer
krexit
*+ (lmd)
* Call system timer vector
* (first guy in the system daisy-chain)
*
*-

        move.w  _timr_ms(a5),-(sp)      ;push #ms/tick
        move.l  etv_timer(a5),a0        ; get vector
        jsr     (a0)                ; call it
        addq    #2,sp               ; cleanup stack

tick1   movem.l (sp)+,d0-d7/a0-a6
t_punt  bclr.b  #5,isrb+mfp         ;clear the interrupt channel
        rte

******************************************
*
*  Quick & dirty sound stuff
*
*
*  Programmed by Dave Staugas
*                14 Mar 1985
*
*
*
*
******************************************
```

```
*
*
*
*
*
*  To start a sound, load the 32-bit address of the
*                       byte stream for that sound in 32-bit
*                       "cursnd", & zero the 8-bit "timer"
*
*
*
*
*   Sound interrupt routine
*   Called from timer C irq
*
sndirq:
        movem.l  a0/d0-d1,-(sp)
        move.l   cursnd(a5),d0              ;get current sound ptr
        beq      snd1                       ;br to exit if zero, inactive
        movea.l  d0,a0                      ;ptr to a0
        move.b   timer(a5),d0               ;check delay timer
        beq.b    snd3                       ;br over delay timer update if not on
*
        subq.b   #1,d0                      ;tick off delay timer
        move.b   d0,timer(a5)               ;save new
        bra.b    snd1                       ;skip sound update this time
snd3:
        move.b   (a0)+,d0                   ;pick up next sound command
        bmi.b    snd2                       ;if minus, go do special
*
        move.b   d0,giselect                ;else, register load command--select this
        cmpi.b   #7,d0                      ;reg. 7 selected?
        bne.b    sn1                        ;br if no
*
        move.b   (a0)+,d1                   ;get data to write to reg 7
        andi.b   #$3f,d1                    ;always leave i/o port settings alone
        move.b   rddata,d0                  ;get mixer contents
        andi.b   #$c0,d0                    ;mask off non-useful info...
        or.b     d1,d0                      ;generate new setting
        move.b   d0,wrdata                  ;write data
        bra.b    snd3                       ;go for next command
sn1:
        move.b   (a0)+,wrdata               ;write next byte as data directly to reg
        bra.b    snd3                       ;go for next command
*
*  special case command
*
snd2:
        addq.b   #1,d0                      ;was command 255?
        bpl.b    snd5                       ;br if yes--set delay timer
*
        cmpi.b   #129,d0                    ;was command 128 (before increment)
        bne.b    snd6                       ;br if not
*
*  command 128
*
```

```
        move.b   (a0)+,auxd(a5)                ;128--set aux data from next byte in strea
        bra.b    snd3                          ;go for next command

  command > 128

nd6.
        cmpi.b   #130,d0                       ;command greater than 129
        bne.b    snd5                          ;br if yes--must be set timer

  command 129

        move.b   (a0)+,giselect                ;129--select register
        move.b   (a0)+,d0                      ;get increment step (signed)
        add.b    d0,auxd(a5)                   ;add to aux data
        move.b   (a0)+,d0                      ;get terminating value
        move.b   auxd(a5),wrdata               ;load reg from data in auxd
        cmp.b    auxd(a5),d0                   ;reached end of cycle?
        beq.b    snd4                          ;br if so

  still within loop, reset sound pointer to iterate for next irq

        subq     #4,a0                         ;back up sound ptr to repeat this command
        bra.b    snd4                          ;update ptr & exit

  set delay timer

nd5:
        move.b   (a0)+,timer(a5)               ;set delay timer from next byter in stream
        bne.b    snd4                          ;if non-zero, real delay here
        movea.w #0,a0                          ;else, sound terminator--set ptr to null
d4:
        move.l   a0,cursnd(a5)                 ;update sound ptr
nd1:
        movem.l (sp)+,a0/d0-d1                 ;pop stack & exit
        rts


  sound data...


  format:

        sound data usually is found in byte pairs, the first of which is the comma
        and the second is the argument.  However, some commands take on more than
        1 argument.

        cmd      function        argument(s)

        00       load reg0       data0
        01       load reg1       data0
        02       load reg2       data0
        03       load reg3       data0
        04       load reg4       data0
        05       load reg5       data0
        06       load reg6       data0
        07       load reg7       data0    note: b7 & b6 forced set for all data to r
```

```
*       08      load reg8       data0
*       09      load reg9       data0
*       0A      load reg10      data0
*       0B      load reg11      data0
*       0C      load reg12      data0
*       0D      load reg13      data0
*
*
*       80      init temp w/    data0
*
*       81      loop defined
*               by 3 args       data0 as register to load using temp
*                               data1 as increment/decrement (signed) of temp
*                               data2 as loop terminator value of temp
*
*       82-FF   set delay
*               timer           data0 is # of counts till next update
*                                       note: if data0 = 0, sound is terminated
*
*
*
*
*
bellsnd:
        .dc.b   0,$34
        .dc.b   1,0
        .dc.b   2,0
        .dc.b   3,0
        .dc.b   4,0
        .dc.b   5,0
        .dc.b   6,0
        .dc.b   7,$FE
        .dc.b   8,$10           ;enable envelope, ch a
        .dc.b   9,0
        .dc.b   10,0
        .dc.b   11,0
        .dc.b   12,$10
        .dc.b   13,9            ;envelope single attack
        .dc.b   255,0
*
keyclk:
        .dc.b   0,$3B
        .dc.b   1,0
        .dc.b   2,0
        .dc.b   3,0
        .dc.b   4,0
        .dc.b   5,0
        .dc.b   6,0
        .dc.b   7,$FE
        .dc.b   8,$10           ;enable envelope, ch a
        .dc.b   13,$3           ;envelope single attack
        .dc.b   11,$80
        .dc.b   12,1
        .dc.b   255,0
*
```

```
*-------------------------------------------------------------------------
*                                                                         :
*       Boot sector                                                       :
*       Loads OS.IMG from the disk and executes it.                       :
*                                                                         :
*       (C)1985 Atari Corp.                                               :
*                                                                         :
* 25-Feb-1985 lmd       Hey!  It fits in 512 bytes....                    :
*  2-Apr-1985 lmd       Fixed bugs (it works now)                         :
*                                                                         :
*-------------------------------------------------------------------------
        text


*
* BPB fields:
*
recsiz  equu    0                       ; size of a sector in bytes
clsiz   equ     2                       ; number of sectors/cluster
clsizb  equ     4                       ; size of a cluster in bytes
rdlen   equ     6                       ; root directory length
fsiz    equ     8                       ; size of a FAT (in sectors)
fatrec  equ     10                      ; start of 2nd FAT
datrec  equ     12                      ; sector# of first data sector
numcl   equ     14                      ; number of clusters on media
bflags  equ     16                      ; flags


*
* OS variables:
*
_membot         equ     $432            ; pointer to bottom of memory
_cmdload        equ     $482            ; load-command switch
bootdev         equ     $446            ; default boot device


*
* Executable code,
* random garbage,
* and a serial number:
*
        bra.s   start                   ; branch to code
        dc.b    'Loader'                ; name of the loader
        dc.b    $00,$00,$00             ; 24-bit serial number

*
* 80 track, single-sided BPB
* (Identical to "DG-1" BPB)
*
        dc.b    $00,$02                 ; #bytes/sector
        dc.b    $02                     ; #sectors/cluster
        dc.b    $01,$00                 ; #reserved sectors
        dc.b    $02                     ; #of FATs
        dc.b    $70,$00                 ; #of root directory entries
        dc.b    $d0,$02                 ; #of sectors on media
        dc.b    $f8                     ; media descriptor byte
        dc.b    $05,$00                         ; #sectors/FAT
        dc.b    $09,$00                         ; #sectors/track
        dc.b    $01,$00                         ; #sides on media
        dc.b    $00,$00                         ; #hidden sectors

        even
*
* Boot parameters
*
execflg:        dc.w    0                       ; copied to _cmdload
ldmode:         dc.w    0                       ; 0:load file, 1:load sectors
ssect:          dc.w    0                       ; starting sector# to load
nsects          dc.w    0                       ; #sectors to load
ldaddr:         dc.l    $40000                  ; load address
fatbuf:         dc.l    $8000                   ; good place for FAT/directory buffer
fname:          dc.b    "OS      IMG"           ; filename to load (11 chars)
*                        12345678901
        even


*+
* Neuter Booter
*
* Register usage:
*       A6 -> FAT buffer
*       A5 -> BPB
*       A4 -> directory/load buffer
*       A3 -> current read address
*       A0..A2 used by traps
*
*       D7 = current cluster number
*       D6 = starting sector/sector number
*       D5 = ending sector
*       D4 = sector count
*       D3 = current sector
*       D0..D2 used by traps
*-
start:
        move.w  execflg(pc),_cmdload    ; set command-load flag

*----- Get BPB for boot device:
        move.w  bootdev,-(sp)           ; d0 = getbpb(bootdev)
        move.w  #7,-(sp)
        trap    #13
        addq    #4,sp
        tst.l   d0                      ; if(d0 == NULL) return;
        beq     _fail                   ; (I give up)
        move.l  d0,a5                   ; a5 -> BPB

        lea     fatbuf(pc),a0           ; if(fatbuf == NULL)
        tst.l   (a0)                    ;       fatbuf = _membot
        bne     fbuf1
        move.l  _membot,(a0)
fbuf1:  move.w  fsiz(a5),d0             ; a4 = fatbuf + (a5[fsiz] << 9)
        lsl.w   #8,d0
        add.l   d0,d0
        move.w  d0,a4
        add.l   fatbuf(pc),a4           ; a4 -> directory buffer

*----- Which mode?
        move.w  ldmode(pc),d0           ; test mode switch
        beq     ldfile                  ; (load file)

*----- Load and exec sectors:
        move.w  ssect(pc),d6            ; starting sector#
        move.w  nsects(pc),d4           ; #sectors to load
        move.l  ldaddr(pc),a3           ; load-address
        bra     l_done                  ; load sectors, execute 'em


*----- Read FAT and directory sectors into memory:
ldfile: move.w  fatrec(a5),d6           ; start = 2nd FAT
        move.w  fsiz(a5),d4             ; count = a5[fsiz] + a5[rdlen]
        add.w   rdlen(a5),d4
        move.l  fatbuf(pc),a3           ; address = the FAT buffer
        bsr     readmult                ; read sectors
        bne     _fail

*----- Setup to search for the image file:
        move.l  a4,a0                   ; a0 -> directory buffer
        move.w  rdlen(a5),d0
        lsl.w   #8,d0
        lsl.w   #1,d0                   ; a0 += rdlen * 512
        lea     (a0,d0.w),a0            ; a0 -> end of directory buffer
        lea     fname(pc),a1            ; a1 -> file to open

*----- Search directory (backwards):
b_3:    sub.w   #$20,a0                 ; backup one directory entry
b_1:    cmp.l   a4,a0                   ; if(a0 < a4) then fail
        blt     _fail                   ; (file not found, so punt)
        moveq   #10,d0                  ; d0 = dbra length of file name
b_2:    move.b  (a0,d0.w),d1            ; compare filename
        cmp.b   (a1,d0.w),d1
        bne     b_3                     ; try next entry on match failure
        dbra    d0,b_2                  ; (try all chars)

*----- Get (byte-reversed) cluster number:
        moveq   #0,d7                   ; get starting cluster number
        move.b  27(a0),d7               ; from byte-reversed entry in
        lsl.w   #8,d7                   ; the directory entry
        move.b  26(a0),d7

*----- Setup for reading the file:
        move.l  fatbuf(pc),a6           ; a6 -> FAT
        move.l  ldaddr(pc),a3           ; a3 -> read address
        clr.l   d4                      ; no sector count

*+
* Read the file.
* Read as many sectors as possible at once (try to suck it
* in with one rwabs call...)
*
```

```
*-


*------ compute sector number from cluster number:
l_1:     cmp.w    #$0ff0,d7               ; end of chain?
         bge      l_done                  ; (yes)

         move.w   d7,d3                   ; d3 = d7 - 2
         subq     #2,d3
         mulu     clsiz(a5),d3            ; d3 *= clsiz
         add.w    datrec(a5),d3           ; d3 += datrec

*------ if "break" in chain of sectors, read some in:
         tst.w    d4                      ; any old sectors?
         beq      l_4                     ; (no)
         cmp.w    d5,d3                   ; can this one be appended?
         beq      l_3                     ; (yes)
         bsr      readmult                ; read old sectors
         bne      _fail                   ; (punt on read failure)

         lsl.l    #8,d4                   ; a3 += count * 512
         lsl.l    #1,d4
         add.l    d4,a3

*------ startup a new chunk of contiguous sectors:
l_4:     move.w   d3,d6                   ; start = current sector
         move.w   d3,d5                   ; end = current sector
         clr.l    d4                      ; count = 0
l_3:     add.w    clsiz(a5),d4            ; append current sector to
         add.w    clsiz(a5),d5            ; the contiguous chunk


*------ compute next cluster number:
         move.w   d7,d2                   ; d2 = (d7 >> 1) + d7
         lsr.w    #1,d2
         add.w    d7,d2
         move.b   1(a6,d2.w),d1           ; get high byte
         lsl.w    #8,d1                   ; shift it up
         move.b   (a6,d2.w),d1            ; get low byte (d1 = raw cluster entry)
         btst     #0,d7                   ; if(d7 & 1) d1 >>= 4
         beq      l_2
         lsr.w    #4,d1
l_2:     and.w    #$0fff,d1               ; d1 &= $0fff
         move.w   d1,d7                   ; d7 = d1
         bra      l_1                     ; read next cluster

*------ read any leftover sectors:
l_done:  tst.w    d4                      ; any sectors left?
         beq      ld_ex                   ; (nothing more to read)
         bsr      readmult                ; read remainder (usu. entire file)
         bne      _fail                   ; (punt on read failure)
ld_ex:   move.l   ldaddr(pc),-(sp)        ; jump to stuff we just loaded
         rts


*------ could not boot: complain
_fail:   clr.l    d0                      ; error = 0
```

```
        rts


*+
* Read sectors from boot device
* Passed:       d6 = logical sector number
*               d4 = count
*               a3 -> address
*
* Returns:      NE: failure
*               EQ: success
*-
readmult:
        move.w  bootdev,-(sp)           ; device = bootdev
        move.w  d6,-(sp)                ; record = d6
        move.w  d4,-(sp)                ; count = d4
        move.l  a3,-(sp)                ; addr = a3
read:   clr.w   -(sp)                   ; operation = READ
        move.w  #4,-(sp)                ; function = rwabs
        trap    #13                     ; bios trap
        add.w   #14,sp                  ; cleanup stack
        tst.w   d0                      ; test return code
        rts


*------
copyrt: dc.b    'Neuter Booter',13,10
        dc.b    '(C)1985 Atari Corp.',13,10
        dc.b    0
```

```
loadable        equ     1                       ; nonzero for loadable driver

*-----------------------------------------------------------------------
*                                                                       :
*       ST SASI hard disk driver                                        :
*       (C)1985 Atari Corp.                                             :
*                                                                       :
*----                                                                   :
*  9-Apr-1985 lmd       Hacked it up.  "Gee, it seems to work ..."      :
* 14-Apr-1985 lmd       linked with BIOS (***FOR NOW***)                :
* 20-Apr-1985 lmd       hacked for WD controller (now, wired...)        :
*                                                                       :
*-----------------------------------------------------------------------


flock           equ     $43e                    ; FIFO lock variable
hdv_init        equ     $46a                    ; hdv_init()
hdv_bpb         equ     $472                    ; hdv_bpb(dev)
hdv_rw          equ     $476                    ; hdv_rw(rw, buf, count, recno, dev)
hdv_boot        equ     $47a                    ; hdv_boot()
hdv_mediach     equ     $47e                    ; hdv_mediach(dev)
_drvbits        equ     $4c2                    ; block device bitVector
_dskbufp        equ     $4c6                    ; pointer to common disk buffer

nretries        equ     3                       ; #retries-1


* --------------- Installer ---------------
        .globl  i_sasi
i_sasi: nop                                     ; stupid assembler

 ifne loadable
        clr.l   -(sp)                           ; it's a bird...
        move.w  #$20,-(sp)                      ;    ... it's a plane ...
        trap    #1                              ;      ... no, its:
        addq    #6,sp                           ; SOOUPERUSER!
        move.l  d0,savssp                       ; "Faster than a prefetched opcode..."
 endc

        bsr     _sasi_init                      ; kick controller
        tst.w   d0
        bmi     isasq                           ; punt -- disk didn't respond

        clr.l   d0
        or.l    _drvbits,d0                     ; include C: bit in devVector
        or.l    #$4,d0
        move.l  d0,_drvbits

        clr.l   a5                              ; zeropage ptr
        move.l  hdv_bpb(a5),o_bpb                       ; save old vectors
        move.l  hdv_rw(a5),o_rw
        move.l  hdv_mediach(a5),o_mediach

        move.l  #hbpb,hdv_bpb(a5)                       ; install our new ones
        move.l  #hrw,hdv_rw(a5)
        move.l  #hmediach,hdv_mediach(a5)
```

```
isasq:  nop                             ; stupid assembler

 ifne loadable
        move.l  savssp,-(sp)            ; become a mild mannered user process
        move.w  #$20,-(sp)
        trap    #1
        addq    #6,sp
 endc
        rts


* ---------------- Front End ----------------


*+
* LONG hbpb(dev) - return ptr to BPB (or NULL)
*
* Passed:       dev     4(sp).W
*
*-
hbpb:
        move.w  4(sp),d0                ; d0 = devno
        move.l  o_bpb,a0                ; a0 -> pass-through vector
        lea     _sasi_bpb(pc),a1        ; a1 -> our handler
        bra     check_dev               ; do it


*+
* LONG rw(rw, buf, count, recno, dev)
*
* Passed:       dev     $e(sp).W
*               recno   $c(sp).W
*               count   $a(sp).W
*               buf     6(sp).L
*               rw      4(sp).W
*
*-
hrw:
        move.w  $e(sp),d0               ; d0 = devno
        move.l  o_rw,a0                 ; a0 -> pass-through vector
        lea     _sasi_rw(pc),a1         ; a1 -> our handler
        bra     check_dev               ; do it


*+
* LONG mediach(dev)
*
* Passed:       dev     4(sp).W
*
*-
hmediach:
        move.w  4(sp),d0                ; d0 = devno
        move.l  o_mediach,a0            ; a0 -> pass-through vector
        lea     _sasi_mediach(pc),a1    ; a1 -> our handler


**+
* check_dev - use handler, or pass vector through
*
* Passed:       d0.w = device#
*               a0 ->  old handler
*               a1 ->  new handler
*               a5 ->  $0000 (zero-page ptr)
*
* Jumps-to:     (a1) if dev in range for this handler
*               (a0) otherwise
*
*-
check_dev:
        cmp.w   #2,d0                   ; devnos match?
        bne     chkd_f                  ; (no)
        move.l  a1,a0                   ; yes -- follow success vector
chkd_f: jmp     (a0)                    ; do it



* ---------------- Medium level driver ----------------

**+
* _sasi_init - initialize SASI dev
* Passed:       nothing
* Returns:      d0 < 0: error
*               d0 ==0: success
*
*-
        .globl  _sasi_init
_sasi_init:

*--- read the boot sector about ten times
        move.w  #9,d7
isas1:  clr.w   -(sp)                   ; dev = 0
        move.l  _dskbufp,-(sp)          ; use disk buffer
        move.w  #1,-(sp)                ; count = 1
        clr.l   -(sp)                   ; sector = 0
        bsr     _hread                  ; read it
        add.w   #12,sp                  ; cleanup stack
        tst.w   d0                      ; test read error return
        dbmi    d7,isas1                ; loop while no error
        bmi     isas2                   ; (punt on error)

        bsr     _wd_setup               ; initialize WD parms
        clr.l   d0
isas2:  rts


**+
* _sasi_bpb - return BPB for hard drive
* Synopsis:     LONG _sasi_bpb(dev)
*               WORD dev;
*
* Returns:      NULL, or a pointer to the BPB buffer
*
*-
        .globl  _sasi_bpb
_sasi_bpb:
        move.l  #thebpb,d0
        rts


*+
* _sasi_rw - read/write hard sectors
* Synopsis:     _sasi_rw(rw, buf, count, recno, dev)
*
* Passed:       dev     $e(sp).W
*               recno   $c(sp).W
*               count   $a(sp).W
*               buf     6(sp).L
*               rw      4(sp).W
*
*-
        .globl  _sasi_rw
_sasi_rw:
        move.w  #nretries,retrycnt      ; setup retry counter

sasrw1: moveq   #0,d0                   ; coerce word to long, unsigned
        move.w  $c(sp),d0               ; sect.L
        move.w  $a(sp),d1               ; count.W
        move.l  6(sp),d2                ; buf.L
        move.w  4(sp),d3                ; rw

        clr.w   -(sp)                   ; dev = 0
        move.l  d2,-(sp)                ; buf
        move.w  d1,-(sp)                ; count
        move.l  d0,-(sp)                ; sect
        tst.w   d3                      ; read or write?
        bne     sasrw3                  ; (write)
        bsr     _hread                  ; read sectors
        bra     sasrw2
sasrw3: bsr     _hwrite                 ; write sectors
sasrw2: add.w   #12,sp                  ; (cleanup stack)
        tst.l   d0                      ; errors?
        beq     sasrwr                  ; no -- success
        subq.w  #1,retrycnt             ; drop retry count and retry
        bpl     sasrw1

sasrwr: rts


*+
* _sasi_mediach - see if hard disk media has changed (it never does)
* Synopsis:     _sasi_mediach(dev)
*               WORD dev;
*
```

```
* Returns:      0L
*
*-
        .globl  _sasi_mediach
_sasi_mediach:
        clr.l   d0
        rts


*+
* BPB for 10MB drive
* Approximate only.  Tweak me.
*
*-
thebpb: dc.w    512                     ; #bytes/sector
        dc.w    2                       ; #sectors/cluster
        dc.w    1024                    ; #bytes/cluster
        dc.w    16                      ; rdlen (256 root files)
        dc.w    41                      ; FATsiz (10300 FAT entries)
        dc.w    42                      ; 2nd FAT start
        dc.w    99                      ; data start
        dc.w    10300                   ; #clusters (approximate here)
        dc.w    1                       ; flags (16-bit FATs)


* ---------------- Low-level driver ----------------


*------ Globals
flock           equu    $43e            ; FIFO lock variable
_hz_200         equ     $4ba            ; 200hz system ticker


*------ Hardware:
wdc             equ     $ff8604
wdl             equ     $ff8606
dmahi           equ     $ff8609
dmamid          equ     dmahi+2
dmalow          equ     dmamid+2
gpip            equ     $fffa01


*------ Tunable:
ltimeout                equ     $10000          ; long-timeout
stimeout                equ     $10000          ; short-timeout


*+
* void _qdone() - Wait for operation complete
* Passed:       nothing
*
* Returns:      EQ: no timeout
*               MI: timeout condition
```

```
*
* Uses:         D0
*-
_qdone:
        move.l  #ltimeout,tocount
qd1:    subq.l  #1,tocount              ; drop timeout count
        bmi     qdq                     ; (i give up, return NE)
        move.b  gpip,d0                 ; interrupt?
        and.b   #$20,d0
        bne     qd1                     ; (not yet)
        move.w  #$80,wdl                ; why do we need to do this
        nop                             ; to the hardware???
        tst.w   wdc
        moveq   #0,d0                   ; return EQ (no timeout)
        rts
qdq:    moveq   #-1,d0                  ; return -1 (error)
        rts


*+
* void _sel()
* Fiddle with SASI lines
*
* Passed:       nothing
*
* Uses:         nothing
*-
_sel:
        move.w  #$88,wdl                ; _FDC + _HDCS + CA1=0(select_latch)
        nop
        move.w  #$20,wdc                ; iomode Rd=data, Wr=controller
        nop
        move.w  #$8a,wdl                ; _FDC + _HDCS + CA1=1(select_io)
        nop
        move.w  #$01,wdc                ; set direction = 1(output)
        nop
        move.w  #$88,wdl                ; _FDC + _HDCS + CA1=0(select_latch)
        nop
        move.w  #$00,wdc                ; iomode Rd=controller, Wr=data
        nop
        move.w  #$8a,wdl                ; _FDC + _HDCS + CA1=1(select_reg)
        nop
        move.w  #$01,wdc                ; write a $01 to data (?)
        rts

*+
* void _reql()
* Wait for /REQ line to go low
*
* Passed:       nothing
*
* Returns:      EQ: ok
*               MI: timeout condition
*
* Uses:         D0
*-
```

```
_reql:
        move.l  #stimeout,tocount       ; setup timeout counter
        move.w  #$88,wdl                ; select SASI status register
        nop
        move.w  #$20,wdc
        nop
        move.w  #$8a,wdl
reql1:  subq.l  #1,tocount              ; drop timeout count
        bmi     reqle                   ; (return NE on timeout)
        move.w  wdc,d0                  ; get SASI status
        and.w   #2,d0                   ; REQ low?
        bne     reql1                   ; (not yet)
        rts
reqle:  moveq   #-1,d0
        rts


*+
* WORD _endcmd()
* Wait for end of SASI command
* Passed:       nothing
*
* Returns:      EQ: success (error code in D0.W)
*               MI: timeout
*               NE: failure (SASI error code in D0.W)
*
* Uses:         D0
*-
_endcmd:
        bsr     _qdone                  ; wait for operation complete
        bmi     endce                   ; (timed-out, so complain)

        move.w  #$88,wdl                ; get completion error code
        nop
        move.w  #$00,wdc
        nop
        move.w  #$8a,wdl
        nop
        move.w  wdc,d1

        bsr     _reql                   ; wait for SASI $00
        bmi     endce                   ; (timeout)

        move.w  #$88,wdl
        nop
        move.w  #$00,wdc
        nop
        move.w  #$8a,wdl
        nop
        tst.w   wdc

        move.w  d1,d0                   ; d0 = error code
        and.w   #$00ff,d0               ; (clean it up)
        rts

endce:  moveq   #-1,d0
```

```
        rts


*+
* _hinit(dev)
* WORD dev;
* Initialize hard disk
*
* Returns:        -1 if hard disk not there
*
*-
_hinit:
        st      flock                   ; lock FIFO
        tst.b   gpip                    ; magic
        bsr     _sel
        moveq   #5,d0
hi_1:   bsr     _reql
        move.w  #$00,wdc
        dbra    d0,hi_1
        bsr     _endcmd
        clr.w   flock                   ; unlock FIFO
        bra     _hdone                  ; cleanup after IRQ


*-
* _hread(sectno, count, buf, dev)
* LONG sectno;            4(sp)
* WORD count;             8(sp)
* LONG buf;              $a(sp)  $b=high, $c=mid, $d=low
* WORD dev;              $e(sp)
*
* Returns:        -1 on timeout
*                 0 on success
*                 nonzero on error
*
*-
        .globl  _hread
_hread:
        st      flock                   ; lock FIFO
        move.l  $a(sp),-(sp)            ; set DMA address
        bsr     _setdma
        addq    #4,sp

        bsr     _sel                    ; select magic
        bmi     _hto
        bsr     _reql                   ; wait for ~REQ
        bmi     _hto
        move.w  #$08,wdc                ; read cmd
        bsr     _reql                   ; ~REQ
        bmi     _hto
        move.b  5(sp),d0                ; construct sector#
        move.b  $e(sp),d1               ; ORed with devno
        lsl.b   #5,d1
        or.b    d1,d0
        move.w  d0,wdc                  ; write MSB sector# + devno
        bsr     _reql                   ; ~REQ
        bmi     _hto
        move.b  6(sp),d0                ; write MidSB sector#
        move.w  d0,wdc
        bsr     _reql                   ; ~REQ
        bmi     _hto
        move.b  7(sp),d0                ; write LSB sector#
        move.w  d0,wdc

        bsr     _reql                   ; write sector count
        bmi     _hto
        move.w  8(sp),wdc

        bsr     _reql
        bmi     _hto
        move.w  #$90,wdl                ; toggle data direction
        nop
        move.w  #$190,wdl
        nop
        move.w  #$90,wdl
        nop
        move.w  8(sp),wdc               ; write sector count to DMA chip
        nop
        move.w  #$8a,wdl
        nop
        move.w  #$07,wdc                ; end-of-command (+fast_step)
        nop
        move.w  #$00,wdl
        bsr     _endcmd
_hrx:   clr.w   flock                   ; unlock FIFO
        bra     _hdone                  ; cleanup after IRQ


*-
* _hwrite(sectno, count, buf, dev)
* LONG sectno;          4(sp)
* WORD count;           8(sp)
* LONG buf;            $a(sp)  $b=high, $c=mid, $d=low
* WORD dev;            $e(sp)
*
*-
        .globl  _hwrite
_hwrite:
        st      flock                   ; lock FIFO

        move.l  $a(sp),-(sp)            ; set DMA address
        bsr     _setdma
        addq    #4,sp

        bsr     _sel
        bmi     _hto
        bsr     _reql
        bmi     _hto
        move.w  #$0a,wdc
        bsr     _reql
        bmi     _hto
        move.b  5(sp),d0
        move.b  $e(sp),d1                       ; ORed with devno
        lsl.b   #5,d1
        or.b    d1,d0
        move.w  d0,wdc
        bsr     _reql
        bmi     _hto
        move.b  6(sp),d0
        move.w  d0,wdc
        bsr     _reql
        bmi     _hto
        move.b  7(sp),d0
        move.w  d0,wdc

        bsr     _reql                           ; sector count
        bmi     _hto
        move.w  8(sp),wdc

        bsr     _reql
        bmi     _hto
        move.w  #$90,wdl
        nop
        move.w  #$190,wdl
        nop
        move.w  8(sp),wdc                       ; sector count
        nop
        move.w  #$18a,wdl
        nop
        move.w  #$07,wdc                        ; end-of-command (+fast_step)
        nop
        move.w  #$100,wdl
        bsr     _endcmd
hwx:    clr.w   flock                           ; unlock FIFO
        bra     _hdone                          ; cleanup after IRQ


*+
* _wd_format - format WD hard disk
* Passed:       nothing
* Returns:      0, or -N
* Uses:         <..?..>
*
*-
        .globl  _wd_format
_wd_format:
        st      flock

        bsr     _sel
        bmi     hfx
        bsr     _reql
        bmi     hfx
        move.w  #4,wdc
        bsr     _reql
        bmi     hfx
        move.w  #0,wdc
        bsr     _reql
        bmi     hfx
        move.w  #0,wdc
        bsr     _reql
        bmi     hfx
        move.w  #0,wdc
        bsr     _reql
        bmi     hfx
        move.w  #0,wdc

        bsr     _reql
        bmi     hfx
        move.w  #$190,wdl
        nop
        move.w  #$90,wdl
        nop
        move.w  #1,wdc
        nop
        move.w  #$8a,wdl
        nop
        move.w  #$07,wdc
        nop
        move.w  #$00,wdl
        bsr     _endcmd
hfx:    clr.w   flock
        bra     _hdone


*+
* _wd_setup - setup parameters for WD hard disk
*
*-
        .globl  _wd_setup
_wd_setup:
        st      flock
        pea     wd_parms(pc)
        bsr     _setdma
        addq    #4,sp

        bsr     _sel
        bmi     wdx
        bsr     _reql
        bmi     wdx
        move.w  #$0c,wdc
        bsr     _reql
        bmi     wdx
        move.w  #$00,wdc
        bsr     _reql
        bmi     wdx
        move.w  #$00,wdc
        bsr     _reql
        bmi     wdx
        move.w  #$00,wdc
        bsr     _reql
        bmi     wdx
        move.w  #$00,wdc
        bsr     _reql
        bmi     wdx
        move.w  #$90,wdl
        nop
        move.w  #$190,wdl
        nop
        move.w  #$01,wdc
        nop
        move.w  #$18a,wdl
        nop
        move.w  #$00,wdc
        nop
        move.w  #$100,wdl
        bsr     _endcmd
wdx:    clr.w   flock
        bra     _hdone

*--- parameters for 10MB WD
wd_parms: dc.b  $02,$64,$02,$01,$31,$01,$31,$0b



*+
* void _setdma(addr)
* LONG addr;
*-
_setdma:
        move.b  7(sp),dmalow
        move.b  6(sp),dmamid
        move.b  5(sp),dmahi
        rts


_hto:   moveq   #-1,d0                          ; indicate timeout
_hdone: move.w  #$80,wdl
        tst.w   wdc
        rts


        bss
savssp:         ds.l    1                       ; (saved SSP)
tocount:        ds.l    1                       ; timeout counter
retrycnt:       ds.w    1                       ; retry counter
o_init:         ds.l    1
o_bpb:          ds.l    1
o_rw:           ds.l    1
o_mediach:      ds.l    1
dma:            ds.l    1                       ; current DMA loc
count:          ds.w    1                       ; current sector count
sect:           ds.l    1                       ; current logical sector
```